(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 1 076 279 A1

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
14.02.2001 Bulletin 2001/07

(51) Int Cl.7: G06F 1/00

(21) Application number: 99306415.3

(22) Date of filing: 13.08.1999

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE
Designated Extension States:
AL LT LV MK RO SI

(71) Applicant: Hewlett-Packard Company
Palo Alto, CA 94304 (US)

(72) Inventor: Pearson, Siani Lynne
Bristol BS9 3PZ (GB)

(74) Representative: Lawman, Matthew John Mitchell
Hewlett-Packard Limited,
IP Section,
Building 3,
Filton Road
Stoke Gifford, Bristol BS34 8QZ (GB)

(54) Computer platforms and their methods of operation

(57) A computer platform (100) uses a tamper-proof component (120), or "trusted module", of a computer platform in conjunction with software, preferably running within the tamper-proof component, that controls the uploading and usage of data on the platform as a generic dongle for that platform. Licensing checks can occur within a trusted environment (in other words, an environment which can be trusted to behave as the user expects); this can be enforced by integrity checking of the uploading and licence-checking software. Metering records can be stored in the tamper-proof device and reported back to administrators as required. There can be an associated clearinghouse mechanism to enable registration and payment for data.

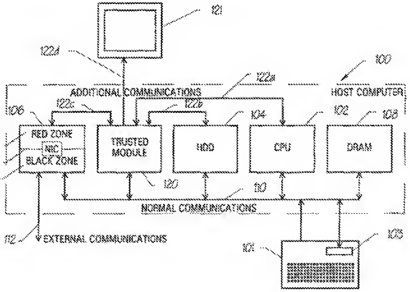


FIG. 14

Printed by Jouve, 75001 PARIS (FR)

Description

[0001] This invention relates to computer platforms and their methods of operation and is more particularly concerned with controlling and/or metering the installation and/or use of data on computer platforms.

[0002] In this specification, 'data' signifies anything that can be formatted digitally, such as images, application software and streaming media. The techniques described in this document can potentially be used to protect or meter many types of information, from simple text documents to audio and video clips, software, graphics, photo- and multimedia materials.

[0003] In the future, computer systems will be able to achieve a more secure booting, together with integrity checks on other code to ensure that viruses or other unauthorised modifications have not been made to the operating systems and mounted software. In addition, a new generation of tamper-proof devices are already appearing or will soon appear on the market and include both external or portable components (such as smart cards) and internal components (embedded processors, semi-embedded processors or co-processors with security functionality, i.e. including motherboard, USB and ISA implementations). These tamper-proof components will be used to check that the hardware of the system has not been tampered with, and to provide a more reliable form of machine identity than currently available (for example, the machine's Ethernet name). Yet how to counteract piracy, and how to licence and meter software in a manner that is acceptable to software developers and end-users will still be a very important problem.

[0004] Software licensing is subject to hackers and piracy, and all the current software licensing methods used have problems associated with them. Software implementations of licensing (such as "licence management systems") are flexible, but not especially secure or fast. In particular, they suffer from a lack of security (for example, being subject to a generic "hack") and difficulty in genuine replacement of software. Conversely, hardware implementations ("dongles") are faster and generally more secure than software implementations, but inflexible. They are tailored only for a particular piece of software and are inconvenient for end-users.

[0005] The present invention, in its preferred embodiment, seeks to deliver the best of both worlds: a hardware implementation that is secure and fast, but with the convenience and flexibility of a software implementation. Increased security in integrity checking on computer platforms, together with more secure key storage, cryptographic capabilities and more secure identification (and hence authentication) within tamper-resistant hardware are provided in the embodiment of this new, generic concept in software licensing and metering.

[0006] A prior patent application (EP 99301100.6) described the use of a Trusted Component to enable verification of the integrity of a computer platform by the reliable measurement and reliable reporting of integrity metrics. This enables the verification of the integrity of a platform by either a local user or a remote entity. That prior patent application described a general method of reporting integrity metrics and verifying the correctness of the integrity of a platform by comparing reported values of metrics with proper values of metrics. The present invention uses licence checking code whose integrity is reported using the method of that prior patent application.

[0007] In overview, the embodiment of the present invention uses a tamper-proof component, or "trusted module" of a computer platform in conjunction with software, preferably running within the tamper-proof component, that controls the uploading and usage of data on the platform as a generic dongle for that platform. Licensing checks can occur within a trusted environment (in other words, an environment which can be trusted to behave as the user expects); this can be enforced by integrity checking of the uploading and licence-checking software. Metering records can be stored in the tamper-proof device and reported back to administrators as required. There can be an associated clearinghouse mechanism to enable registration and payment for data.

[0008] More formally, in accordance with a first aspect of the present invention, there is provided a computer platform having: a trusted module which is resistant to internal tampering and which stores a third party's public key certificate; means storing licence-related code comprising at least one of: a secure executor (which is preferably generic) for checking whether the platform or a user thereof is licensed to use particular data and for providing an interface for using the data and/or for monitoring its usage; and a secure loader (which is preferably generic) for checking whether the platform or a user thereof is licensed to install particular data and/or for checking for data integrity before installation; and means storing a hashed version of the licence-related code signed with the third party's private key; wherein the computer platform is programmed so that, upon booting of the platform, the licence-related code is integrity checked with reference to the signed version and the public key certificate; and if the integrity check fails, the licence-related code is prevented from being loaded. If the integrity check fails, it may be arranged that the complete platform integrity fails.

[0009] In the context of this specification, the term "user" includes may mean an end user of the platform, or a system administrator, or both.

[0010] The trusted module or component, as described in the prior patent application mentioned above is preferably immune to unauthorised modification or inspection of internal data. It is physical to prevent forgery, tamper-resistant to prevent counterfeiting, and preferably has crypto functions to securely communicate at a distance. Methods of building trusted modules are, per se, well known to those skilled in the art. The trusted

module may use cryptographic methods to give itself a cryptographic identity and to provide authenticity, integrity, confidentiality, guard against replay attacks, make digital signatures, and use digital certificates as required. These and other crypto methods and their initialisation are well known to those skilled in the art of security.

[0011] Preferably, the integrity checking is performed by reading and hashing the licence-related code to produce a first hash; reading and decrypting the signed version using the public key certificate to produce a second hash, and comparing the first and second hashes.

[0012] Preferably, the licence-related code also includes secure key-transfer code for enabling a licence key to be transferred between the trusted module and a further trusted module of another computer platform. This key transfer code is particularly useful in improving key management when using licensing models that involve an unlock key, that is, where the data is transmitted in an encrypted form and the unlock key is used to allow the protected data to be decrypted and run. The transfer may be carried out by using a public key infrastructure to encrypt a message containing an unlock key, and checking for integrity via hashing and digital signatures. There may be an option to transfer the data itself in this manner, using the secure loader.

[0013] Preferably, the licence-related code also includes a library of interface subroutines which can be called in order to communicate with the trusted module. The client library is a collection of high-level interface subroutines that applications call to communicate with the trusted module. The client library may also be used by software executors (see below) for communication with the trusted module and operating system ('OS').

[0014] The licence-related code may include, for at least one group of data, a (or a respective) software executor which specifies the respective group of data and which is operable to act as an interface to that group of data. This allows methods of licensing protection specific to the protected data, and therefore potentially a greater level of protection. If a software executor is associated with an application, optionally it processes queries (API calls) submitted by the application.

[0015] Preferably, if space permits, the means storing the licence-related code and/or the means storing the hashed version of the licence-related code are provided, at least in part, by the trusted module.

[0016] Preferably, the trusted module and an operating system of the platform have a dedicated communications path therebetween which is inaccessible to other parts of the computer platform.

[0017] Next the way in which these components interact to form a system for general-purpose data licensing will be considered. There are several stages in which such a system can be constructed, which may be considered as progressing from each other. The first stage is to improve upon current licensing methods such as dongles to make the trusted module act as a generic dongle, governed by generic licence-related software (as detailed above) that performs licence checking and is protected against bypassing by integrity checking. Such licence-checking software need not run within the trusted module itself. A preferred stage is the logical extension of such a system in which the licensing software runs within the trusted module. A request to load or execute some data will be sent to the trusted module, preferably from the software executor. The licensing software in the trusted module will evaluate such a request and decide whether to allow this, based on details of licensing rights. If the request is to be allowed, this information is conveyed to the OS via a hardware communications path from the trusted module to the CPU. The communications path is preferably inaccessible to ordinary applications and non-OS software. The OS then starts the process to load or execute the data, as appropriate.

[0018] Various methods are now considered in which the system components may interact to perform useful licensing functionality. First consideration is given to the way in which the secure loader operates to install data.

[0019] In one installation mode: the operating system is operable to request the secure loader to licence-check whether the platform or a user thereof (e.g. an end user or a system administrator) is licensed to install that particular data and/or to check the integrity of that data; in response to such a request, the secure loader is operable to perform such a check and respond to the operating system with the result of the check, and in dependence upon the response, the operating system is operable to install or not to install the particular data. This check on the platform or user may be performed by various methods, such as checking for the presence of a private application key or other secret in the trusted module or in a smart card, or checking for the identity and presence of the trusted module or smart card. Such an identity could be made known to the developer, or such a secret could be inserted into the trusted module or smart card during a registration process. This is analogous to the process which will be described later in Example A.

[0020] In this mode, preferably the operating system is programmed to install the particular data only in response to the secure loader. Also, in this mode, preferably: the trusted module stores a public key certificate for a party associated with the particular data to be installed; the operating system is operable to include, in the request to check, the particular data together with a hashed version thereof signed with a private key of the associated party; in performing the check, the secure loader is operable: to hash the particular data included in the request to produce a third hash; to decrypt the signed hashed version in the request using the public key certificate for the associated party to produce a fourth hash; and to generate the response in dependence upon whether or not the third and fourth hashes match.

[0021] This checks for integrity of the message. The integrity checking mechanism also prevents replay attacks by using a standard mechanism, such as challenge/response, or introducing a history of the communications in the hash. The problem of non-repudiation can be avoided by keeping private keys in tamper proof hardware. Preferably, the request to check includes the software executor for the particular data.

[0022] In another installation mode, the software executor (or at least one of the software executors) is operable to request the trusted module to install particular data; in response to such a request, the secure loader within the trusted module is operable to licence-check whether the platform or a user thereof is licensed to install that particular data and/or to check the integrity of that data and to respond to the operating system with the result of the check; and in dependence upon the response, the operating system is operable to install or not to install the particular data.

[0023] The check may be carried out in a similar fashion to that described above in relation to said one installation mode.

[0024] In this other mode, preferably the operating system is programmed to install the particular data only in response to the trusted module. Also, in this mode, preferably the response from the trusted module to the operating system is supplied via the dedicated communications path, as described above.

[0025] With either of these installation modes, if the check succeeds, the trusted module is preferably operable to generate a log for auditing the particular data. Also, if the check succeeds, the secure loader is preferably operable to perform a virus check on the particular data.

[0026] Upon installation, the particular data may be installed into the trusted platform. Alternatively, the platform may include a further, removable, trusted module (such as a smart card) and be operable to perform an authentication check between the first-mentioned trusted module and the removable trusted module, in which case, upon installation, the particular data may be installed into the further trusted module.

[0027] The software executor may itself be protected via integrity checks, carried out by the secure loader. For example, this procedure may work as follows.

(a) The software executor is customised such that the public key corresponding to the client's trusted module is included within it.

(b) The data, associated with a customised software executor, is sent to the client.

(c) Both the data and the software executor are hashed and signed with the clearinghouse/developer's private key, and this is sent in conjunction with the data and software executor.

(d) The secure loader integrity checks the software executor when it is received - upon installation of the software executor, the package is verified by hashing and comparison with the decrypted signature (using the public key in the trusted module). The software executor is not loaded if the digital signature does not match what is expected, and in this case the secure loader signals an error. The secure loader also integrity checks the data itself, using the same procedure.

[0028] Now consideration is given to the way in which the secure executor operates to use data.

[0029] In a first execution mode, the software executor (or at least one of the software executors) contains a public key of the trusted module and a licensing model for the respective data; the operating system is operable to request that software executor that its respective data be used; in response to such a request, that software executor is operable to request the secure executor to licence-check, using its licensing model, whether the platform or a user thereof is licensed to use that data; in response to such latter request, the secure executor is operable to perform the requested licence-check, to sign the result of the licence check using a private key of the trusted module, and to respond to that software executor with the signed result; in response to such a response, that software executor is operable: to check the integrity of the signed result using the public key of the trusted module; and upon a successful integrity check of a successful licence-check result, to request the operating system to use that data.

[0030] In a second execution mode, the software executor (or at least one of the software executors) contains a public key of the trusted module and a licensing model for the respective data, the operating system is operable to request the secure executor that particular data be used; in response to such a request, the secure executor is operable to send to the respective software executor a request, signed using a private key of the trusted module, for a licensing model for the particular data; in response to such latter request, that software executor is operable: to check the integrity of the request using the public key of the trusted module; and upon a successful integrity check, to send the licensing model to the secure executor; and upon receipt of the licensing model, the secure executor is operable: to perform a licence-check using that licensing model; and upon a successful licence-check, to request the operating system to use that data.

[0031] In a third execution mode, the secure executor contains at least one licensing model; the operating system is operable to request the secure executor that particular data be used; and in response to such a request, the secure executor is operable to perform a licence-check using the, or one of the, licensing models; and upon a successful licence-check, to request the operating system to use that data.

[0032] With any of these three execution modes, preferably the operating system is programmed to use the particular data only in response to the secure executor or the software executor.

[0033] In a fourth execution mode: the secure executor contains at least one licensing model; the software executor (or at least one of the software executors) is operable to request the trusted module that its respective data be used; in response to such a request, the secure executor within the trusted module is operable to perform a licence-check using the, or one of the, licensing models; and upon a successful licence-check, to request the operating system to use that data. In this case, preferably, the operating system is programmed to use the particular data only in response to the trusted module.

[0034] With any of the second to fourth execution modes, the request from the secure executor to the operating system to use the data is preferably supplied via the dedicated communications path.

[0035] With any of the first to fourth execution modes, preferably the trusted module is operable to log the request to the operating system to use the data. The security and reliability of licensing or metering is enhanced by securely logging data usage within the trusted module. Logging of licensing-related activity is carried out and recorded securely in the tamper-proof component. There is the option to carry this out at a number of different stages during licensing. The most common would be at the stage at which the data was allowed to run by the secure executor or software executor. Another common point would be at the stage at which the secure loader has successfully completed its integrity checks on the data to be installed, and has successfully installed this data onto the client machine. Since the secure executor, software executor and secure loader are protected by integrity checks, some protection is given against hackers trying to bypass or edit the logging process. Such logs would provide both secure auditing information and the possibility of flexible licensing and payment models such as pay-per-use, renting, time-dependent charges, and so on. Such audit logs would form the basis for usage reports and information accessible to third parties such as the machine user's IT department or company auditors. They would also have commercial value, such as for advertising or giving feedback on ratings.

[0036] In the case where the platform includes a further, removable, trusted module (such as a smart card) as mentioned above, it preferably includes a user identity, and, upon licence-checking the secure executor or software executor is operable to perform the licence-check with reference to the user identity.

[0037] When the user asks to run software or access protected data, the secure executor can perform the licence-check, for example, by:

(a) Checking for a secret corresponding to a software or data reference, in a device, or

(b) Using an unlock key to decrypt data and allowing it to execute (there are various options for differing functionality of the unlock key, including partial unlocking of the code), or

(c) Checking for licensing rights in a database, corresponding to a data reference and a device identity, or

(d) Retrieving a key from a database, corresponding to a data reference and a device identity, and using this to unlock the data.

[0038] When the user tries to run an application, it may be arranged that the secure executor assumes overall control, and that it retrieves information from the software executor, if one is present, associated with the data to find out which specific check is preferred by the developer. If a type of check is specified, the secure executor will carry this out; otherwise it will use a default check, as described below. If the check succeeds, the secure executor will execute the data. If the check fails, the secure executor will prevent the data from being executed.

[0039] If the software executor does not specify a licensing method, or there is no software executor attached to the application, the secure executor may use a default protocol that will have been defined for the particular machine. This will have been set by the machine's administrator with the machine's environment in mind; for example, if the machine is only used by one person, a licensing model corresponding to the internal trusted module would probably be most appropriate. It will not be possible to bypass the secure executor, and hence the licensing checks, because the secure executor code will have been included within the platform integrity check as part of the boot integrity procedure.

[0040] Different models of licensing use the secure executor and software executor in different ways. As will be appreciated from the above, it is possible to use them in combination, or with either performing the licensing checks. There are two main preferred options:

(1) The first option is to have different software executors attached to each piece of data, governing licence checking within the secure executor for those particular pieces of data. In some of the examples in the next section, the software executors communicate directly with the operating system in this way.

(2) An alternative approach is to place more emphasis upon the secure executor, by building up the generic code within the platform which carries out the checks, and having the secure executor act as a bridge between the OS and any software executors.

This alternative avoids putting the burden of the protocol-writing on the developer, allows the developer to specify licensing choices very easily and makes use of integrity checking of licence checking code when the platform integrity check is made.

[0041] The software executor associated with a piece of data may include any particular information to be checked for (obtained during the registration process) together with information notifying the secure executor within the computer platform about the method of licensing to be used, the particular trusted device on which to make the check, and a reference to the data which is to be protected. For example, *licensing_method(secret, sc,k,w)* and *licensing_method(secret,tc,k,w)* indicate that the software referenced by w should be allowed to run on a machine only if the secret *k* is found stored within the current smart card or internal trusted component, respectively, of the machine.

[0042] Different software executors are attached to data, with software executors indicating which type of licensing model is to be used. The secure executor carries out a check at runtime, according to this licensing model, and does not allow the software w to run unless the check succeeds. By these means, communication from the clearinghouse to the trusted module specifies which licensing protocol the clearinghouse wishes to use.

[0043] Various specific protocols may be employed by the secure executor. For example, in a first protocol:

- the secure executor checks the trusted module ID entry or smart card ID entry;
- optionally, the secure executor downloads database entries into a profile stored within the trusted module;
- the secure executor checks in an external database or a profile stored within the trusted module against a data reference and the trusted module ID entry (or smart card ID entry) for an unlock key for the data;
- the secure executor retrieves this key and decrypts the associated data so that it may be executed by the operating system;
- optionally, the secure executor stores the unlock key within the trusted module, along with the data reference;
- the data is protected via encryption or partial encryption using the corresponding key;
- there are various options for differing functionality of the unlock key, and
- in return for payment, the database entry corresponding to the trusted module ID will be updated with this key.

[0044] In a second protocol:

- optionally, the secure executor downloads database entries into a profile stored within the trusted module,
- the secure executor checks in an external database or a profile stored within the trusted module for licensing rights, corresponding to a data reference and the trusted module ID entry (or smart card ID entry);
- only if there are appropriate licensing rights, the secure executor authorises the OS to execute the data; and
- in return for payment, the database entry corresponding to the trusted module ID or smart card ID will be updated with an appropriate permission.

[0045] In a third protocol:

- the secure executor checks for a secret corresponding to a software or data reference in a trusted module (including a smart card);
- the secret to be checked for is specified by the software executor associated with the data whose licence is being checked; and
- only if the secret is present in the trusted module will the secure executor authorise the OS to execute the associated software or data.

[0046] In a fourth protocol:

- the secure executor uses an unlock key associated with some data stored within the trusted module or smart card to decrypt the data so that it may be executed by the operating system; and
- there are various options for differing functionality of the unlock key, including partial unlocking of the code.

[0047] In a fifth protocol:

- the secure executor uses a key associated with some data stored within the trusted module or smart card, or else inputted from the end-user via the keyboard, the trusted module or smart card ID and a pre-defined algorithm to calculate a decryption key;
- the secure executor uses the decryption key to de-

crypt the data so that it may be executed by the operating system;

- there are various options for differing functionality of the decryption key, including partial unlocking of the code.

[0048] In a sixth protocol:

- the secure executor allows use of floating licences for a group of users;

- the secure executor checks in a database against the trusted module ID or smart card ID entry for a licence key for the data;

- the secure executor retrieves a licence key (if one were available) in order to allow that particular execution; and

- the secure executor returns the licence key to the pool when the data execution is closed.

[0049] In a seventh protocol:

- the secure executor performs a combination of any the first to sixth protocols, such that different methods of licence checking can be used for different data entities;

- the choice of protocol can be determined by the secure executor itself;

- a default or overriding protocol can be defined by an administrator; and

- the protocol to be used when checking licensing for particular data is determined by any software executor associated with that data.

[0050] Some licensing models later described in this document do not prevent copying of data, but just inhibit unauthorised use of data and secure the logging of usage on machines that have the tamper-proof device as part of the platform. The desired level of data protection depends upon the business model. Data can be sent via traditional and other non-secure channels. However, it is most important that the licence key transfer is secure.

[0051] In accordance with a second aspect of the present invention, there is provided a method of transferring a licence (or a key therefor) from a first to a second computer platform each in accordance with the first aspect of the invention, the method comprising the steps of: setting up secure communication between the trusted modules; sending the licence or the key therefor from the first trusted module to the second trusted module using the secure communication; and deleting the licence or the key therefor from the first trusted module.

[0052] There are many situations in which a customer might wish to transfer a licence to another person or to another machine. For example, if a new PC were purchased, if software is upgraded or replaced, or if the customer wishes to run an application on a portable instead of a desktop. Moving a hardware dongle specific to each application is the easy solution and there is the analogous solution of using specific smart cards. However, all systems which provide a generic dongle, and therefore are more practical in most situations for end-users, are faced with a major problem of key management in this situation. Wave System's WaveNet and licence management systems ('LMFs') are no exception. Software-only methods require an installation/deinstallation process, or else have to trust the end user to use only the number of licences legitimately purchased when a second password is issued for the same licence.

[0053] The options for licence transferral using trusted modules depend upon the licensing aspect that is adopted. In general, these are as follows:

[0054] For licensing using a database check, the database entries corresponding to both machine trusted module IDs (if the licence is changed to another machine) or both smart card IDs (if the licence is changed to another person) should be changed.

[0055] For licensing involving a trusted module related finger-print check or using code tailored to the trusted module, the new device (i.e. a smart card, if changing a licence to another person; the internal trusted module, if changing a licence to another machine) should be re-registered with the vendor, and another key or tailored software issued based on the new device ID obtained respectively.

[0056] For methods involving encryption and an unlock key, if there is one smart card per application, the appropriate smart card (and any pins) should be given to the new licensee. Otherwise, the unlock key and data can be transferred between trusted modules automatically, without the need for the vendor to be involved beyond receiving a report of the transfer (as described in the eighth method). This involves integrity checking of associated data, copying a licence key from one trusted module to another, and deinstalling the licence from the original trusted module.

[0057] The stages in transferring a licence (i.e. unlock key L) for data S from TC1 in client machine M1 to TC2 in machine M2 are, for example, as follows:

A. Secure key transfer code ('SKT') is integrity checked as an extension of the BIS procedure. The licence transfer code is hashed and signed with the manufacturer's private key. Upon boot/installation of the platform, the package is verified by hashing, and comparison with the decrypted signature to check integrity, using a public key certificate installed into the trusted module by the manufacturer. The licence transfer code will not be loaded if the digital signature does not match what is expected,

and the platform integrity check will fail.

B. Initialisation. The content provider already has the public key of TC1 via the original registration and data installation process. If not, this is sent to him.

1. If the owner of TC1 wishes to transfer the licence to TC2, there is a call from the OS of machine M1 to the SKT within M1 to transfer the licence for data S to TC2.

2. SKT in M1 generates a random number R and sends a message to M2 asking for the licence to be transferred, containing a reference to the data S, together with the public key certificate of TC1.

3. If M2 obtains authorisation from an appropriate source, SKT in M2 replies in the affirmative, including R, the public key certificate of TC2, a reference to S, and a new nonce T that it has generated.

4. SKT in M1 then sends to M2 the public key certificate of the content provider of S, together with T.

These communications are appended to a hashed version of the communication signed by the trusted module's private key in the sender's machine, so that the receiver SKT can check the integrity of the message. If the integrity checks fail, messages are sent by each SKT to the OS within their machines and the protocol stops.

C. Program upload. If the above authentication is successful, TC1 hashes the data S (optionally a version already signed by the content provider) and signs it with the private key of TC1 (for example, using Microsoft's Authenticode). TC1 then uploads this signature together with the data into TC2. Optionally, the data is encrypted.

D. Code verification. The secure loader within TC2 verifies the signatures of the data S: it checks the signature using TC1's public key, thereby retrieving the message hash; next it computes the hash of S to check that it is the same as the decrypted message hash. If this validation is successful, the secure loader installs the program into the machine corresponding to TC2. If not, it generates an error message to the SKT which blocks further passage of the licence transfer protocol.

E. Transfer key. The SKT in M1 generates a symmetric key using a random number generator, and uses it to encrypt a message transporting the unlock key. The SKT in M1 sends this message to the SKT in M2, together with the symmetric key encrypted with TC2's public key and also a hash of all this information, signed with TC1's private key. Only TC2 will have the RSA private key to decrypt the symmetric key, which will allow decryption of the unlock key.

F. Message verification. The SKT in M2 checks the signature using the public key of TC1, and decrypts the message using the symmetric key obtained by decryption using TC2's private key, thus obtaining the unlock key. If the signature is correct, the key is stored within the trusted component, and associated with the data S. If the signature is not correct, an error message is sent to the SKT in M1 and the protocol stops.

G. Key deleted from TC1, and content provider notified. The SKT in M1 deletes the unlock key from TC1 and makes a log of this in TC1. The SKT in M1 sends a message to the content provider, signed using the private key of TC1, informing the content provider that the licence for code S has been transferred to M2. Optionally, SKT in M1 or in M2 sends a message to the data vendor giving details of how the owner of M2 may be contacted for registration.

[0058] There is an option for the trusted component, and the software executor, to act as a new part of the operating system, and form a bridge between the operating system and applications, by providing an environment for certain functions. For example, API calls can be made to the trusted module such as 'save' and 'restore'. 'Save' will pass data through the trusted module, which will encrypt the data in the trusted module and store it either in the trusted module or on the hard disk. It will not be possible to access this data without the permission of the trusted module. There is an additional option to carry out some transformations within the trusted module using such data, and for the software to use API calls to request information from the trusted module and get an answer exported. In summary, API calls can be used from the software executor or application code to the trusted module to check the presence of the trusted module or a private application key stored on the trusted module (analogous to existing dongle methods), and further, to use the trusted module for providing an environment for certain functions or data storage.

[0059] More specifically, API calls may be added to the application code or the software executor and used to query the OS, trusted module or secure executor via the client library. For example, API calls may be added to the application code or the software executor and used to query the trusted module or secure executor via the client library to check for the presence of a private application key or other secret in the trusted module or smart card or to check for the identity and presence of

the trusted module or smart card.

[0060] In one particular model which will be described in more detail later, a licensing model is employed in which an entry in a licensing-related database corresponding to the trusted module's ID is updated, and the secure executor will only allow data to run once permissions on this database have been checked. In this case, the software executor associated with an application calls the secure executor (possibly in the trusted module), the secure executor checks the licensing rights, and if this check succeeds, passes the call to the operating system ("OS") in order for the application to be run in the normal manner. In other words, the OS accepts calls to execute data only if the call comes from secure licence-related code such as the secure executor or software executor.

[0061] In another particular model which will be described in more detail later, the trusted module preferably stores hardware and/or software used to implement the invention and the OS accepts calls to execute data if the call comes from the trusted module. In particular, the trusted module preferably acts as a bridge between an application and the OS and the OS preferably ignores all requests to load applications except for those from the trusted module.

[0062] One possible licensing model would be for the secure executor to check in a database against the trusted module ID entry for an unlock key for the data. In this case the data is protected via encryption or partial encryption using the corresponding key, and hence can be freely distributed without fear of piracy. Once payment is made, the database entry corresponding to the trusted module's ID will be updated with this key. When the user wishes to run the application, the key can be retrieved to allow the data to be unlocked. The key may then be stored in the tamper-proof device so that the database look-up need only happen once. However, in licensing models where floating licences are desired, it would be more appropriate to store such keys centrally and allow access only on each execution, so that the licence can then be restored to the appropriate group for use by another user. Thus, a model for licence "exchange" is provided.

[0063] Accordingly, the present invention extends to the case in which there is optional interaction between the secure executor, the software executor and the trusted module to use floating licences for a group of users via the secure executor or software executor instigating a check in a database against the trusted module ID entry for a licence key for the software, retrieving a licence key (if one were available) in order to allow that particular execution, and returning the licence key to the pool when the application is closed.

[0064] In order to accommodate more flexible situations such as hot-desking, when a variety of users use generic terminals, a combination of multiple trusted devices can be used. In particular, a combination of fixed tamper-proof components and portable tamper-proof components gives great flexibility in licensing. Most obviously, a personal user's smart card would be used in combination with an internal tamper-proof device within the computer. On this type of licensing model, the software executor or secure executor would run the data only if a particular smart card is present (or one of a selected group of smart cards is present).

[0065] The internal trusted module contains a trusted machine identity, and the portable trusted module (in this case, a smart card) contains an identity specific to the user (which could be authenticated using an incorporated biometric device). Many different ways of licensing could be used in such a situation (one example is given in the following section), and these are analogous to the options presented in the 'Preferred Embodiment' section. The differences are that, according to the particular model implemented:

- The smart card identity is involved in the licensing check carried out by the secure executor or software executor, rather than the internal machine identity. Hence, for example, the user identity is checked against the profile or directory rather than the machine identity. In the case of unlock keys being stored on the smart card, the presence of the smart card ID within the trusted module will cause the secure executor when requiring the unlock key to (a) copy the unlock key in an encrypted form to the trusted module, by the smart card encrypting it using the trusted module's public key, or (b) use the unlock key from the smart card directly.
- There is authentication between the internal trusted module and the smart card. Authentication between the smart card and trusted module is carried out at the stage at which the smart card is inserted, and the current smart card ID is temporarily stored within the trusted module, to be used for the licensing check in the same way as the trusted module ID would have been used in the licensing models described in this document (see Examples A, B and F described later). When the smart card is removed, or (with single sign on) the user logs out, this temporary smart card ID value within the trusted module is reset to a null value.

[0066] Both user-based licensing and machine-based licensing could be used for different data within the same machine. This could be done by (a) checking directory entries against the smart card ID rather than the machine ID if the smart card ID value within the trusted module is not null (and against the machine ID if this fails), or (b) checking for an unlock key within the smart card if a smart card is currently inserted in the reader - that is to say, either requesting this to be copied to the trusted module, or using it directly.

[0067] Accordingly, the invention extends to the case in which there is optional use of a combination of an in-

ternal machine trusted module and a portable trusted module (and the secure executor and software executor) to perform licence checking based on the user identity associated with the portable trusted module.

[0068] A licensing system of the present invention which will be described in more detail below, has the following features:

- the computer platform is registered with a third party C. Optionally, C is given the trusted module ID or smart card ID;
- authentication between the trusted module and C and exchange of public key certificates takes place before, or at the same time as, exchange of DES session keys for confidentiality of the messages;
- the secure loader performs an integrity check on the data, and only installs the data if this succeeds;
- the data is executed using one of the protocols described above; and
- each developer can use either generic or specific content protection.

[0069] In one form:

- data encrypted using a key K is signed under C's private code signing key and sent by C to the trusted module;
- the unlock key corresponding to K is encrypted by C using the trusted module's public key, signed using C's private code signing key, and sent to the computer platform; and
- the key transfer code decrypts the unlock key, checks integrity and the signature, and this key is then stored in the trusted module, associated with the relevant data.

[0070] In another form:

- data encrypted using a key K is signed under C's private code signing key and sent by C to the trusted module;
- an unlock key is transferred from C to the end-user of the computer platform or to the computer platform;
- the key transfer code calculates the decryption key corresponding to K from the unlock key, the trusted module or smart card ID and a pre-stored algorithm;
- optionally, the previous stage is carried out by the secure executor or software executor associated with the data; and
- this decryption key is then stored in the trusted module or a smart card, associated with the relevant data.

[0071] In a further form:

- data encrypted using a key K and any associated software executor is signed under C's private code signing key and sent by C to the trusted module; and
- the unlock key corresponding to K is inserted into the database entry corresponding to the trusted module ID or smart card ID.

[0072] In yet another form:

- data and any associated software executor is signed under C's private code signing key and sent by C to the trusted module; and
- an entry corresponding to permission to execute the data is inserted into the database entry corresponding to the trusted module ID or smart card ID, or vice versa.

[0073] A specific embodiment of the present invention will now be described, purely by way of example, with reference to the accompanying drawings, in which:

Figure 1	is a diagram which shows the motherboard of computing apparatus adapted to include a trusted device and as described in the prior patent application mentioned above;
Figure 2	is a diagram which shows in more detail the trusted device shown in Figure 1;
Figure 3	is a diagram which shows in the contents of a certificate stored in the trusted device;
Figure 4	is a diagram, which shows the features of a measurement function responsible for acquiring an integrity metric;
Figure 5	is a flow diagram which illustrates the steps involved in acquiring an integrity metric of the computing apparatus;
Figure 6	is a flow diagram which illustrates the steps involved in establishing

communications between a trusted computing platform and a remote platform including the trusted platform verifying its integrity;

Figure 7	is a schematic block diagram of a host computer system which is the subject of another patent application (applicant's ref. 30990086) having the same filing date as the present application,
Figure 8	is a schematic block diagram of a trusted module in the system of Figure 7,
Figures 9 to 12	show parts of the system of Figure 7 to illustrate various communication methods employed therein;
Figure 13	illustrates the format of a protocol data unit used in the system of Figure 7;
Figure 14	shows a modification to the system of Figure 7, which will be used to describe a specific embodiment of the present invention;
Figure 15	is a diagram of the logical components of a trusted module in the system of Figure 14;
Figure 16	illustrates the structure of protected software of data in the system of Figure 14;
Figure 17	is a flow chart illustrating installing or upgrading software or other data on the system of Figure 14;
Figure 18	is a flow chart illustrating the use of protected software or data in the system of Figure 14 employing one model of licence checking;
Figure 19	is a flow chart illustrating the use of protected software or data in the system of Figure 14 employing another model of licence checking; and
Figure 20	is a flow chart illustrating the use of protected software or data in the system of Figure 14 employing a further model of licence checking.

[0074] Before describing the embodiment of the present invention, the computing platform incorporating a trusted device which is the subject of the prior patent application mentioned above will firstly be described with reference to Figures 1 to 6.

[0075] That application describes the incorporation into a computing platform of a physical trusted device or module whose function is to bind the identity of the platform to reliably measured data that provides an integrity metric of the platform. The identity and the integrity metric are compared with expected values provided by a trusted party (TP) that is prepared to vouch for the trustworthiness of the platform. If there is a match, the implication is that at least part of the platform is operating correctly, depending on the scope of the integrity metric.

[0076] A user verifies the correct operation of the platform before exchanging other data with the platform. A user does this by requesting the trusted device to provide its identity and an integrity metric. (Optionally the trusted device will refuse to provide evidence of identity if it itself was unable to verify correct operation of the platform.) The user receives the proof of identity and the integrity metric, and compares them against values which it believes to be true. Those proper values are provided by the TP or another entity that is trusted by the user. If data reported by the trusted device is the same as that provided by the TP, the user trusts the platform. This is because the user trusts the entity. The entity trusts the platform because it has previously validated the identity and determined the proper integrity metric of the platform.

[0077] Once a user has established trusted operation of the platform, he exchanges other data with the platform. For a local user, the exchange might be by interacting with some software application running on the platform. For a remote user, the exchange might involve a secure transaction. In either case, the data exchanged is 'signed' by the trusted device. The user can then have greater confidence that data is being exchanged with a platform whose behaviour can be trusted.

[0078] The trusted device uses cryptographic processes but does not necessarily provide an external interface to those cryptographic processes. Also, a most desirable implementation would be to make the trusted device tamperproof, to protect secrets by making them inaccessible to other platform functions and provide an environment that is substantially immune to unauthorised modification. Since tamper-proofing is impossible, the best approximation is a trusted device that is tamper-resistant, or tamper-detecting. The trusted device, therefore, preferably consists of one physical component that is tamper-resistant.

[0079] Techniques relevant to tamper-resistance are well known to those skilled in the art of security. These techniques include methods for resisting tampering, methods for detecting tampering, and methods for eliminating data when tampering is detected. It will be appreciated that, although tamper-proofing is a most de-

sirable feature of the present invention, it does not enter into the normal operation of the invention and, as such, is beyond the scope of the present invention and will not be described in any detail herein.

[0080] The trusted device is preferably a physical one because it must be difficult to forge. It is most preferably tamper-resistant because it must be hard to counterfeit. It typically has an engine capable of using cryptographic processes because it is required to prove identity, both locally and at a distance, and it contains at least one method of measuring some integrity metric of the platform with which it is associated.

[0081] Figure 1 illustrates the motherboard 10 of an exemplary computer platform (not shown). The motherboard 10 includes (among other standard components) a main processor 11, main memory 12, a trusted device 14, a data bus 16 and respective standard control lines 17 and address lines 18, and BIOS memory 19 containing the BIOS program for the platform.

[0082] Typically, the BIOS program is located in a special reserved memory area, the upper 64K of the first megabyte do the system memory (addresses FØØØh to FFFFh), and the main processor is arranged to look at this memory location first, in accordance with an industry wide standard.

[0083] The significant difference between the platform and a conventional platform is that, after reset, the main processor is initially controlled by the trusted device, which then hands control over to the platform-specific BIOS program, which in turn initialises all input/output devices as normal. After the BIOS program has executed, control is handed over as normal by the BIOS program to an operating system program, such as Windows NT (TM), which is typically loaded into main memory 12 from a hard disk drive (not shown).

[0084] Clearly, this change from the normal procedure requires a modification to the implementation of the industry standard, whereby the main processor 11 is directed to address the trusted device 14 to receive its first instructions. This change may be made simply by hard-coding a different address into the main processor 11. Alternatively, the trusted device 14 may be assigned the standard BIOS program address, in which case there is no need to modify the main processor configuration.

[0085] Although, the trusted device 14 is a described as a single, discrete component, it is envisaged that the functions of the trusted device 14 may alternatively be split into multiple devices on the motherboard, or even integrated into one or more of the existing standard devices of the platform. For example, it is feasible to integrate one or more of the functions of the trusted device into the main processor itself, provided that the functions and their communications cannot be subverted. This, however, would probably require separate leads on the processor for sole use by the trusted functions. Additionally or alternatively, although the trusted device is described as a hardware device that is adapted for integration into the motherboard 10, it is anticipated that a trusted device may be implemented as a 'removable' device, such as a dongle, which could be attached to a platform when required. Whether the trusted device is integrated or removable is a matter of design choice.

[0086] The trusted device 14 comprises a number of blocks, as illustrated in Figure 2: a controller 20 for controlling the overall operation of the trusted device 14, and interacting with the other functions on the trusted device 14 and with the other devices on the motherboard 10; a measurement function 21 for acquiring an integrity metric from the platform; a cryptographic function 22 for signing or encrypting specified data, and interface circuitry 23 having appropriate ports (24, 25 & 26) for connecting the trusted device 14 respectively to the data bus 16, control lines 17 and address lines 18 of the motherboard 10. Each of the blocks in the trusted device 14 has access (typically via the controller 20) to appropriate volatile memory areas 27 and/or non-volatile memory areas 28 of the trusted device 14.

[0087] For reasons of performance, the trusted device 14 may be implemented as an application specific integrated circuit (ASIC). However, for flexibility, the trusted device is preferably an appropriately programmed micro-controller. Both ASICs and micro-controllers are well known in the art of microelectronics and will not be considered herein in any further detail.

[0088] One item of data stored in the non-volatile memory is a certificate 30, which is illustrated in Figure 3. The certificate 30 contains at least a public key 32 of the trusted device 14 and an authenticated value of a platform integrity metric 34 measured by a TP. Optionally, the trusted device 14 also contains an identity (ID) label 36 of the trusted device 14.

[0089] Where present, the ID label 36 is a conventional ID label, for example a serial number, that is unique within some context. The ID label 36 is generally used for indexing and labelling of data relevant to the trusted device 14, but is insufficient in itself to prove the identity of the platform under trusted conditions.

[0090] The trusted device 14 is equipped with at least one method of reliably measuring some integrity metric of the computing platform with which it is associated. The integrity metric is acquired by the measurement function 21, which is illustrated in more detail in Figure 4.

[0091] The measurement function 21 has access to non-volatile memory 40 for storing a hash program 41, plus volatile memory 42 for storing a computed integrity metric 43, in the form of a digest. The hash program 41 contains instructions for computing the digest, in code that is native to the main processor 11. In addition, part of the measurement function 21 is configured to respond to the main processor 11 as if it were addressable memory, such as standard read-only memory, by sensing memory read signals addressed to the trusted device 14 and returning appropriate data. The result is that the main processor 11 sees the trusted device, for the purposes of integrity metric measurement, as a standard read-only memory.

[0092] In the preferred implementation, as well as the digest, the integrity metric includes a Boolean value 44, which is stored in volatile memory 45 by the measurement function 21, for reasons that will become apparent.

[0093] A preferred process for acquiring an integrity metric will now be described with reference to Figure 5.

[0094] In step 500, at switch-on, the measurement function 21 monitors the activity of the main processor 11 on the data, control and address lines (16, 17 & 18) to determine whether the trusted device 14 is the first memory accessed. Under conventional operation, a main processor would first be directed to the BIOS memory first in order to execute the BIOS program. However, in accordance with the present embodiment, the main processor 11 is directed to the trusted device 14, which acts as a memory. In step 505, if the trusted device 14 is the first memory accessed, in step 510, the measurement function 21 writes to volatile memory 45 a Boolean value 44, which indicates that the trusted device 14 was the first memory accessed. Otherwise, in step 515, the measurement function writes a Boolean value 44, which indicates that the trusted device 14 was not the first memory accessed.

[0095] In the event the trusted device 14 is not the first accessed, there is of course a chance that the trusted device 14 will not be accessed at all. This would be the case, for example, if the main processor 11 were manipulated to run the BIOS program first. Under these circumstances, the platform would operate, but would be unable to verify its integrity on demand, since the integrity metric would not be available. Further, if the trusted device 14 were accessed after the BIOS program had been accessed, the Boolean value 44 would clearly indicate lack of integrity of the platform.

[0096] In step 520, when (or if) accessed as a memory by the main processor 11, the main processor 11 reads the stored native hash instructions 41 from the measurement function 21 in step 525. The hash instructions 41 are passed for processing by the main processor 11 over the data bus 16. In step 530, main processor 11 executes the hash instructions 41 and uses them, in step 535, to compute a digest of the BIOS memory 19, by reading the contents of the BIOS memory 19 and processing those contents according to the hash program. In step 540, the main processor 11 writes the computed digest 43 to the appropriate non-volatile memory location 42 in the trusted device 14. The measurement function 21, in step 545, then calls the BIOS program in the BIOS memory 19, and execution continues in a conventional manner.

[0097] Clearly, there are a number of different ways in which the integrity metric may be calculated, depending upon the scope of the trust required. The measurement of the BIOS program's integrity provides a fundamental check on the integrity of a platform's underlying processing environment. Other integrity checks could involve establishing that various other devices, components or apparatus attached to the platform are present and in correct working order. In one example, the BIOS programs associated with a SCSI controller could be verified to ensure communications with peripheral equipment could be trusted. In another example, the integrity of other devices, for example memory devices or co-processors, on the platform could be verified by enacting fixed challenge/response interactions to ensure consistent results. Also, although in the present embodiment the trusted device 14 utilises the data bus as its main means of communication with other parts of the platform, it would be feasible, although not so convenient, to provide alternative communications paths, such as hard-wired paths or optical paths. Further, although in the present embodiment the trusted device 14 instructs the main processor 11 to calculate the integrity metric, it is anticipated that, in other embodiments, the trusted device itself will be arranged to measure one or more integrity metrics.

[0098] Preferably, the BIOS boot process includes mechanisms to verify the integrity of the boot process itself. Such mechanisms are already known from, for example, Intel's draft "Wired for Management baseline specification v 2.0 - BOOT Integrity Service", and involve calculating digests of software or firmware before loading that software or firmware. Such a computed digest is compared with a value stored in a certificate provided by a trusted entity, whose public key is known to the BIOS. The software/firmware is then loaded only if the computed value matches the expected value from the certificate, and the certificate has been proven valid by use of the trusted entity's public key. Otherwise, an appropriate exception handling routine is invoked.

[0099] Optionally, after receiving the computed BIOS digest, the trusted device 14 may inspect the proper value of the BIOS digest in the certificate and not pass control to the BIOS if the computed digest does not match the proper value. Additionally, or alternatively, the trusted device 14 may inspect the Boolean value 44 and not pass control back to the BIOS if the trusted device 14 was not the first memory accessed.

[0100] Figure 6 illustrates the flow of actions by a TP, the trusted device 14 incorporated into a platform, and a user (of a remote platform) who wants to verify the integrity of the trusted platform. It will be appreciated that substantially the same steps as are depicted in Figure 6 are involved when the user is a local user. In either case, the user would typically rely on some form of software application to enact the verification. It would be possible to run the software application on the remote platform or the trusted platform. However, there is a chance that, even on the remote platform, the software application could be subverted in some way. Therefore, it is anticipated that, for a high level of integrity, the software application would reside on a smart card of the user, who would insert the smart card into an appropriate reader for the purposes of verification.

[0101] At the first instance, a TP, which vouches for trusted platforms, will inspect the type of the platform to

decide whether to vouch for it or not. This will be a matter of policy. If all is well, in step 600, the TP measures the value of integrity metric of the platform. Then, the TP generates a certificate, in step 605, for the platform. The certificate is generated by the TP by appending the trusted device's public key, and optionally its ID label, to the measured integrity metric, and signing the string with the TP's private key.

[0102] The trusted device 14 can subsequently prove its identity by using its private key to process some input data received from the user and produce output data, such that the input/output pair is statistically impossible to produce without knowledge of the private key. Hence, knowledge of the private key forms the basis of identity in this case. Clearly, it would be feasible to use symmetric encryption to form the basis of identity. However, the disadvantage of using symmetric encryption is that the user would need to share his secret with the trusted device. Further, as a result of the need to share the secret with the user, while symmetric encryption would in principle be sufficient to prove identity to the user, it would insufficient to prove identity to a third party, who could not be entirely sure the verification originated from the trusted device or the user.

[0103] In step 610, the trusted device 14 is initialised by writing the certificate 30 into the appropriate non-volatile memory locations of the trusted device 14. This is done, preferably, by secure communication with the trusted device 14 after it is installed in the motherboard 10. The method of writing the certificate to the trusted device 14 is analogous to the method used to initialise smart cards by writing private keys thereto. The secure communications is supported by a 'master key', known only to the TP, that is written to the trusted device (or smart card) during manufacture, and used to enable the writing of data to the trusted device 14; writing of data to the trusted device 14 without knowledge of the master key is not possible.

[0104] At some later point during operation of the platform, for example when it is switched on or reset, in step 615, the trusted device 14 acquires and stores the integrity metric 43 of the platform.

[0105] When a user wishes to communicate with the platform, in step 620, he creates a nonce, such as a random number, and, in step 625, challenges the trusted device 14 (the operating system of the platform, or an appropriate software application, is arranged to recognise the challenge and pass it to the trusted device 14, typically via a BIOS-type call, in an appropriate fashion). The nonce is used to protect the user from deception caused by replay of old but genuine signatures (called a 'replay attack') by untrustworthy platforms. The process of providing a nonce and verifying the response is an example of the well-known 'challenge/response' process.

[0106] In step 630, the trusted device 14 receives the challenge and creates a digest of the measured integrity metric and the nonce, and optionally its ID label. Then, in step 635, the trusted device 14 signs the digest, using its private key, and returns the signed digest, accompanied by the certificate 30, to the user.

[0107] In step 640, the user receives the challenge response and verifies the certificate using the well known public key of the TP. The user then, in step 650, extracts the trusted device's 14 public key from the certificate and uses it to decrypt the signed digest from the challenge response. Then, in step 660, the user verifies the nonce inside the challenge response. Next, in step 670, the user compares the computed integrity metric, which it extracts from the challenge response, with the proper platform integrity metric, which it extracts from the certificate. If any of the foregoing verification steps fails, in steps 645, 655, 665 or 675, the whole process ends in step 680 with no further communications taking place.

[0108] Assuming all is well, in steps 685 and 690, the user and the trusted platform use other protocols to set up secure communications for other data, where the data from the platform is preferably signed by the trusted device 14.

[0109] The techniques of signing, using certificates, and challenge/response, and using them to prove identity, are well known to those skilled in the art of security and will, thus, not be described in any more detail herein.

[0110] Referring now to Figures 7 to 13, a specific embodiment of the invention which is the subject of the other patent application, mentioned above, having the same filing date as the present application will now be described. In Figure 7, a host computer 100 has a main CPU 102, a hard disk drive 104, a PCI network interface card 106 and DRAM memory 108 with conventional ("normal") communications paths 110 (such as ISA, EISA, PCI, USB) therebetween. The network interface card 106 also has an external communication path 112 with the world outside the host computer 100.

[0111] The network interface card 106 is logically divided into "red" and "black" data zones 114,116 with an interface 118 therebetween. In the red zone 114, data is usually plain text and is sensitive and vulnerable to undetectable alteration and undesired eavesdropping. In the black data zone 116, data is protected from undetected alteration and undesired eavesdropping (preferably encrypted by standard crypto mechanisms). The interface 118 ensures that red information does not leak into the black zone 116. The interface 118 preferably uses standard crypto methods and electronic isolation techniques to separate the red and black zones 114,116. The design and construction of such red and black zones 114,116 and the interface 118 is well known to those skilled in the art of security and electronics, particularly in the military field. The normal communication path 110 and external communication path 112 connect with the black zone 116 of the network interface card 106.

[0112] The host computer 100 also includes a trusted module 120 which is connected, not only to the normal communication paths 110, but also by mutually separate

additional communication paths 122 (sub-referenced 122a,122b,122c) to the CPU 102, hard disk drive 104 and the red zone 114 of the network interface card 106. By way of example, the trusted module 120 does not have such a separate additional communication path 122 with the memory 108.

[0113] The trusted module 120 can communicate with the CPU 102, hard disk drive 104 and red zone 114 of the network interface card 106 via the additional communication paths 122a,b,c, respectively. It can also communicate with the CPU 102, hard disk drive 104, black zone 116 of the network interface card 106 and the memory 108 via the normal communication paths 110. The trusted module 120 can also act as a 100VG switching centre to route certain information between the CPU 102, hard disk drive 104 and the red zone 114 of the network interface card 106, *via* the trusted module 120 and the additional communication paths 122, under control of a policy stored in the trusted module. The trusted module 120 can also generate cryptographic keys and distribute those keys to the CPU 102, the hard disk drive 104, and the red zone 114 of the network interface card 106 via the additional communication paths 122a,b,c, respectively.

[0114] Figure 8 illustrates the physical architecture of the trusted module 120. A first switching engine 124 is connected separately to the additional communication paths 122a,b,c and also to an internal communication path 126 of the trusted module 120. This switching engine 124 is under control of a policy loaded into the trusted module 120. Other components of the trusted module 120 are:

* a computing engine 128 that manages the trusted module 120 and performs general purpose computing for the trusted module 120;
* volatile memory 130 that stores temporary data;
* non-volatile memory 132 that stores long term data;
* cryptographic engines 134 that perform specialist crypto functions such as encryption and key generation;
* a random number source 136 used primarily in crypto operations;
* a second switching engine 138 that connects the trusted module 120 to the normal communication paths 110; and
* tamper detection mechanisms 140,

all connected to the internal communication path 126 of the trusted module 120.

[0115] The trusted module 120 is based on a trusted device or module 14 as described in more detail above with reference to Figures 1 to 6.

[0116] With regard to crypto key generation and distribution, the trusted module 120 generates cryptographic keys, using the random number generator 136, a hash algorithm, and other algorithms, all of which are well known, *per se*, to those skilled in the art of security. The trusted module 120 distributes selected keys to the CPU 102, hard disk drive 104 and the red zone 114 of the network interface card 106 using the additional communication paths 122a,b,c, respectively, rather than the normal communications paths 110. Keys may be used for communications between the internal modules 102,104,106,120 of the platform over the normal communication paths 110. Other temporary keys may be used (by the network interface card 106 or CPU 102) for bulk encryption or decryption of external data using the SSL protocol after the trusted module 120 has completed the SSL handshaking phase that uses long term identity secrets that must not be revealed outside the trusted module 120. Other temporary keys may be used (by the hard disk drive 104 or CPU 102) for bulk encryption or decryption of data stored on the hard disk drive 104 after those temporary keys have been created or revealed inside the trusted module 120 using long term secrets that must not be revealed outside the trusted module 120.

[0117] The trusted module 120 enforces policy control over communications between modules by the selective distribution of encryption keys. The trusted module 120 enforces a policy ban on communications between given pairs of modules by refusing to issue keys that enable secure communications over the shared infrastructure 110 between those pairs of modules.

[0118] Figure 9 illustrates a process by which the trusted module 120 can perform a watchdog function and 'ping' the modules 102,104,106 connected to the additional communication paths 122. The trusted module generates a challenge 142 and sends it to the CPU 102, hard disk drive 104 and red zone 114 of the network interface card 106 using the additional communication paths 122a,b,c, respectively. Each of the CPU 102, hard disk drive 104 and network interface card 106 responds with a response 144a,b,c, respectively, on the respective additional communication path 122a,b,c to say whether the respective module is active, and preferably that the module is acting properly. The trusted module 120 notes the responses 144a,b,c and uses them as metrics in its responses to integrity challenges that are described above with reference to Figures 1 to 6.

[0119] Figure 10 illustrates the process by which incoming external secure messages are processed when the trusted module 120 is the only module in the platform with cryptographic capabilities. An external message 146 is received by the black zone 116 of the network interface card 106 using the external communication path 112. The network interface card 106 sends a protocol data unit 148 (to be described in further detail later) containing some data and a request for an authentication and integrity check to the trusted module 120 using the normal communication paths 110. The trusted module 120 performs the authentication and integrity checks using the long term keys inside the trusted module 120 that must not revealed outside the trusted module 120, and sends a protocol data unit 150 containing an 'OK'

indication to the red zone 114 of the network interface card 106 using the additional communication path 122c. The network interface card 106 then sends a protocol data unit 152 containing some data and a request for decryption to the trusted module 120 using the normal communication paths 110. The trusted module 120 decrypts the data using either temporary or long term keys inside the trusted module 120, and sends a protocol data unit 154 containing the decrypted data to the CPU 102 using the additional communication path 122a. The CPU then takes appropriate action.

[0120] Figure 11 illustrates the process by which the CPU 102 requests a policy decision from the trusted module 120. This could be used, for example, when the CPU 102 must determine whether policy allows certain data to be manipulated or an application to be executed. This will be described in more later with reference to Figures 14 to 20. The CPU 102 sends a protocol data unit 156 containing a request to the trusted module 120 using the normal communication paths 110. The trusted module 120 processes the request 156 according to the policy stored inside the trusted module 120. The trusted module 120 sends a protocol data unit 158 containing a reply to the CPU 102 using the additional communication path 122a, in order that the CPU 102 can be sure that authorisation came from the trusted module 120. If the action is authorised, the CPU 102 takes the necessary action. Otherwise, it abandons the process.

[0121] Figure 12 illustrates an example of the control of policy over protected communications between the modules 102,104,106. All of the communications in this example use the additional communication paths 122. The red zone 114 of the network interface card 106 sends a protocol data unit 160 that is destined for the hard disk drive 104 to the trusted module 120 on the additional data path 122c. In the case where the policy does not permit this, the trusted module 120 denies the request by sending a protocol data unit 162 containing a denial to the network interface card 106 on the additional data path 122c. Later, the CPU 102 requests sensitive data from the hard disk drive 104 by sending a protocol data unit 164 addressed to the hard disk drive, but sent on the additional data path 122a to the trusted module 120. The trusted module 120 checks that the policy allows this. In the case where it does, the trusted module 120 relays the protocol data unit 164 to the hard disk drive 104 on the additional data path 122b. The hard disk drive 104 provides the data and sends it in a protocol data unit 166 on the additional data path 122b back to the trusted module 120 addressed to the CPU 102. The trusted module 120 checks that the policy allows this, and, in the case where it does, relays the protocol data unit 166 to the CPU 102 on the additional data path 122a.

[0122] Figure 13 illustrates the format of the data protocol units 178 by which data is passed over the additional communication paths 122. The data protocol unit 178 has:-

- an identifier field 168 indicating the type of the protocol data unit;
- a length field 170 indicating the length of the protocol data unit;
- a source field 172 indicating the source of the protocol data unit;
- a destination field 174 indicating the destination of the protocol data unit;
- and so on, including in many cases a data field 176.

[0123] Not all fields are always necessary. For example, assuming the policy of the trusted module 120 forbids it to relay key protocol data units that that did not originate within the trusted module 120, the CPU 102, hard disk drive 104 and network interface card 106 can therefore assume that keys are always from the trusted module 120. Hence, source and destination fields are unnecessary in key protocol data units - such protocol data units are implicitly authenticated. The design and construction and use, *per se*, of protocol data units is well known to those skilled in the art of communications.

[0124] The specific embodiment of the present invention will now be described with reference to Figures 14 to 20. Figure 14 illustrates the physical system and is a development of the system described above with reference to Figures 7 to 13. In Figure 14, a display 121 is connected to the trusted module 120 by means of one 122d of the additional communications paths as described above. This enables the trusted module 120 to reliably write to the display, without fear of subversion from normal software, including the operating system. Also, the host computer 100 is connected to a keyboard 101 that has a built-in smart card reader 103, both of which are connected to the normal communications paths 110. A smart card which is inserted into the smart card reader 103 can be considered to be an additional trusted module and is therefore able to communicate securely with the trusted module 120.

[0125] Figure 15 illustrates a logical diagram of the components of the trusted module 120, comprising licensing code components 200 and other licensing data components 202 within the trusted module 120. The licensing code components 200 run within a protected environment, as previously described, and preferably within the trusted module 120 itself, and comprise: a secure executor 204, a secure loader 206, secure key-transfer code 208 and a client library 210. The licence-related data components 202 stored on the trusted module 120 include the private key 212 of the trusted module 120, the public key certificate 214 of a trusted entity, the clearinghouse or developer's public key certificate 216, a licensing log 218, and a hashed version 220 of the licence-related code 200, signed with the private key of the trusted entity who has the public key certificate 214.

[0126] Figure 16 illustrates the structure of protected software or data 222 within the client computer 100. Digital data 224 on the client computer 100 is associated with a respective software executor 226, within which is

stored the public key 228 of the trusted module 120. This structure 230 is stored together with a hashed version 232 of it, signed with the clearinghouse or developer's private key. There will be a structure analogous to the resulting unit 222 for each piece of protected software or data.

[0127] Figure 17 illustrates the flowchart for loading or upgrading software or other data onto the client platform, for the general case where the secure loader 206 may not be running within the trusted module 120.

[0128] The data to be installed is hashed and signed with the sender's private key, and this is appended to the data itself by the sender.

[0129] In step 234, the operating system sends a request, together with the data and the signed hashed version, to the secure loader 206 that the data be installed. In step 236, the secure loader 206 receives the request, and in step 238 it checks the signature of this message, using the public key certificate corresponding to the sender, thereby checking authentication of the sender.

[0130] If authentication fails, then in step 240 the secure loader 206 sends an error message to the operating system. In step 242 the operating system receives this error message, and in step 244 displays an appropriate message.

[0131] If authentication succeeds in step 238, then in step 246 the secure loader 206 computes the hash of the message, via the cryptographic capabilities available within the trusted module 120, and in step 248 compares it to the message hash that is associated with the data and was received in step 236. This checks for integrity of the message.

[0132] If the hashes are not the same, this indicates that the data has been altered, and that it should not be installed. In this case, in step 250 the secure loader 206 sends an error message to the OS, which then performs steps 242,244 described above.

[0133] If the hashes are found to be the same in step 248, then in step 252 the trusted module 120 makes a log of the installation, and in step 254 the secure loader 206 indicates to the OS that the data can be installed as normal, which then happens in step 256.

[0134] If other forms of check (particularly licence checks) are additionally or alternatively to be employed, these may be included between steps 250 and 252 in the method described with reference to Figure 17.

[0135] Figure 18 illustrates the flowchart for licensing using a model of licence checking where the OS communicates with the secure executor 204, and the software executor 226 associated with a piece of data has the option to choose the licensing model to be used for protection of that data. This again is for the general case where licensing software is not necessarily mounted within the trusted module 120. The procedure is as follows:

[0136] When the user wishes to run some digital data, in step 258 a request is sent by the operating system, which is received by the secure executor 204 in step 260. In step 262, the secure executor 206 generates a random number (nonce), and in step 264 issues a challenge/response to the software executor 226 corresponding to that piece of data, by means of sending the nonce, together with a reference to the application (e.g. its title), signed using the private key 212 of the trusted module 120.

[0137] Following receipt in step 266 by the software executor 226, in step 268 it verifies and authenticates the secure executor's challenge using the public key 228 of the trusted module 120. If there is an error, or if the software executor 226 does not wish the data to be executed on this particular machine, an error message is sent in step 270, which is relayed by the secure executor 204 in step 272 to the operating system. Following receipt of such an error message in step 274, the operating system displays an appropriate error message in step 276 and the data is not executed.

[0138] If there is no error in step 268, then in step 278 the software executor 226 returns a message to the secure executor 204 incorporating the nonce, the reference to the data and optionally a licensing model. The nonce is included to give protection against replay attacks.

[0139] Having received the message in step 280, then in step 282 the secure executor 204 makes the appropriate licensing check dependent upon the licensing model specified by the software executor. This may involve unlocking the data using a key. Further details of these licensing models are considered later. If there is no software executor associated with the data, the secure executor makes a licensing check corresponding to a default licensing model previously set within it by an administrator. If there is a valid licence, in step 284 the secure executor 204 asks the trusted module 120 to takes a metering record of the transaction, steps 286,288, and in step 290 sends permission to the operating system to execute the data. Upon receipt in step 292, the operating system executes the data in step 294. Following the licensing check in step 282, if there is no valid licence, in step 296 the secure executor 204 asks the operating system to notify the end-user appropriately, steps 274,276, and the data is not executed.

[0140] Figure 19 is a flowchart for licensing using a model of licence checking where the OS communicates with the software executors 226 rather than the secure executor 204. This again is for the general case where licensing software is not necessarily mounted within the trusted module 120.

[0141] When the user wishes to execute some data, in step 298 the OS sends a message to the software executor 226 associated with the data, received in step 300. In step 302, the software executor 226 generates a random number (nonce), and in step 304 issues a challenge/response to the secure executor 204 within the trusted module 120, by means of sending the nonce, together with a reference to the data. In addition, a smart card ID is sent, if that was used to log in to the client

machine and hot-desking is the licensing model to be used.

[0142] Following receipt in step 306 of the message, in step 308 the secure executor 204 makes an appropriate licensing check on the data. If there is no valid licence, then in step 310 the secure executor 204 returns an error message, from which the software executor could determine the exact type of problem with licensing and notifies the OS appropriately, steps 312,314,316.

[0143] If there is a valid licence, then in step 318 the secure executor 204 returns a message incorporating the nonce and reference to the data, signed and encrypted using the private key 212 of the trusted module 120. The nonce is included to give protection against replay attacks.

[0144] Following receipt in step 320 of the message, in step 322 the software executor 226 verifies if the secure executor's reply is correct using the public key certificate 228 of the trusted module 120. If it is correct, then in step 324 the software executor 226 asks trusted module 120 makes a log, steps 326,328 and in step 330 passes the call to the OS to execute the data, steps 332,334. On the other hand, if it is not correct, in step 336 the software executor 226 sends an error message to the OS, which then displays an error message as appropriate, steps 314,316.

[0145] In a preferred mechanism for enforcing checks on permission to execute digital data, the trusted module 120 includes the hardware and/or stores the software used to implement the invention. In particular, the trusted module 120 acts as a bridge between an application and the OS. The OS preferably ignores all requests to load or run applications except those from the trusted module 120, given via a communications path 122 between the trusted module 120 and the CPU 102 that is preferably inaccessible to ordinary applications and non-OS software. The processes operating on the host computer are as follows. First, there is an initial request to the trusted module 120 to execute an application or other data, preferably via the software executor 226 associated with this data, and usually in response to some action by the end-user. The software executor 226 will contain the public key certificate 228 of the trusted module 120 on which the data is installed or to be installed. The secure executor 204 within the trusted module 120 will carry out appropriate licence checking, as detailed above. If the result of this checking is that it is appropriate to execute the data, the secure executor 204 will convey this information to the OS via a communications path 122 to the CPU 102, which is preferably inaccessible to ordinary applications and non-OS software. The OS then starts a process on the host to execute the application or data. An analogous process will be carried out when the secure loader communicates with the OS to indicate that data installation is appropriate, or when the key transfer code communicates with the OS to transfer unlock keys.

[0146] Figure 20 illustrates the flowchart for licensing using a model of licence checking as mentioned above, where licensing software is stored within the trusted module 120, and the trusted module 120 acts as a bridge between an application and the OS. The process is similar to that given in Figure 19, except that the secure executor 204 is within the trusted module 120 itself and the secure executor 120 uses a communication path 122 (preferably dedicated) from the trusted module 120 to the CPU 102 when communicating with the OS.

[0147] There are many different ways in which this invention can be used. Details of six of these will now be presented.

Example A

[0148] A first example is to use tamper-resistant hardware as a generic dongle by binding applications to the hardware. Major differences between this example and the other examples in this section are firstly that licensing protection is carried out when the code is actually executing, and secondly that this method is suited to protection of applications for which source code is available to the party carrying out the protection mechanism.

[0149] Software is loaded into the platform (and optionally into the tamper-resistant hardware, where it would be run). The software is integrity checked using the secure loader. API calls are used to the trusted module to check for the presence of a secret in the trusted module or check for the identity and presence of the trusted module. In addition, the trusted module can be made to execute part of the code. Strong authentication of the trusted module is possible by using the trusted module's private cryptographic key, and standard authentication protocols.

[0150] In addition, there are the following options:

- API calls can be made to the trusted module instead of the OS (as discussed earlier).
- The trusted module can be made to execute part of the code. This can be done in several ways, some of which have already been discussed.
- Part of the code could be marked for transferral into tamper-resistant hardware (such as the internal trusted module or a smart card), where it may be stored in an encrypted form, and calls made to this functionality elsewhere within the code.
- Analogously, portable trusted modules such as smart cards can be made to execute part of the code.

[0151] The use of this method rather than the analogous use of API calls to a hardware dongle counters many of the disadvantages normally associated with this approach.

[0152] First, traditional software protection using API calls to a hardware dongle is vulnerable to modification of software locks via a debugger (for example, by stepping through communications between processors and the motherboard) or disassembler, thus altering the code to remove calls to the key. Modified copies of the code are produced, and run freely, both on the host machine and on other machines. This may be countered in this method by:

* Part of the code being run within the trusted module itself
* Integrity checks on the platform and associated software that ensure that associated licence-checking code must be loaded together with the software, and prevent licence checks from being bypassed.

[0153] Secondly, there is a danger currently that record and playback (or other techniques) could be used to fill in some of the missing functionality of processing carried out on hardware. This is countered in this method by integrity checks on the software and on licence-checking code.

[0154] Thirdly, there is much greater flexibility in the licensing model, both in that the licence need not be tied to the machine, and in the greater choice of payment models. The trusted module provides a generic dongle that is not just tailored to a specific application and in addition provides greater capacity for licensing information storage, and better metering.

[0155] Finally, there are effort-related gains for the developer. The benefits of addition of API calls to the software are that the software is customised for a particular machine, and hence not immediately of benefit on another machine, even if the executable or source code were obtained in clear. However, it can require substantial effort on the part of the developer. By the only difference being a different trusted module ID, with protection via integrity-checking of code, substantial protection can be gained with very little effort by the developer. Again, running part of the code within the trusted module itself does not require individual customisation of code.

[0156] In this example:-

* The developer can do any combination of the following.
  * Insert API calls into the software, and/or into a software executor associated with the software. These will check
    * for the presence of a secret in the tamper-resistant device (e.g. if the developer has made smart card dongles and shipped these to the end users), or
    * for the identity and presence of a tamper-proof device within the end-user's machine (using this as a generic dongle).

    A software executor will generally only make a check at runtime; further API calls within the code can be made at various stages during execution of the code if desired. This is done in a general way for the software (i.e. each customer will receive the same version), and customised details such as the exact trusted module ID can be added later, at the registration stage described below.
  * Insert a secret into the software executor associated with the data, together with information notifying the secure executor within the computer platform that the licensing method of using a check for the presence of a secret in the trusted module or some other trusted device is to be used. For example, licensing_method(secret,sc,k,w) or licensing_method(secret,tc,k,w) indicates that the software referenced by w should only be allowed to run on a machine if the secret k is found stored within the current smart card or internal trusted component of the machine. The secure executor will have a protocol pre-stored that allows it to carry out this check, and will not allow the software w to run unless the check succeeds.
* The user registers with the developer. As part of the initialisation process, authentication between communicating parties within the licensing system will take place before (or at the same time, by the protocols being incorporated) as exchange of session keys for confidentiality of messages passed between them (see example B for further details of this process). The tamper-proof component is sent public-key certificates corresponding to the developer. In return for payment (1) he is given the generally customised software, together with a portable hardware-resistant device (such as a smart card) containing (by storage or hard-coding) the developer's secret that is checked for in the code, or a key is transferred to his tamper-proof device (for example, by an analogous method to that described in more detail in example B below, except that this key is not an unlock key for decryption of the software) (2) his machine ID is inserted into the software (in order that API calls check for that particular machine ID) and the software is shipped to him.
* In order to control interactions between the application and trusted module, the developer needs to ship two additional components to customers, namely the software executor and client library. The client library is a collection of high-level interface subroutines that the application calls to communi-

cate with the software executor.

- The software and the code described in the previous two stages above are signed by using a hashed version of the message signed by the sender's private key appended to the message, so that the receiver can check the integrity of the message. More explicitly, the developer hashes the code M, and signs it with his private key (Sprk) to produce the signature $\Sigma_{Sprk}(h(M))$. Then he sends this signature together with the message M.

- The secure loader will then check the signature, using the developer's public key, and therefore retrieve the message hash. This guarantees that the sender is the one whose public key has been used to check the signature. Having the message, and the message hash, the secure loader can then compute the hash of the message and compare it to the message hash it has decrypted. This checks for integrity of the message. Furthermore the integrity checking mechanism should prevent replay-attacks by some standard mechanism - such as challenge/response, or introducing a history of the communications in the hash.

- If the integrity check works, the secure loader installs the software. This ensures that modified software (e.g. without API calls) cannot be run, viruses are not introduced, etc. The software can also be modified to check for the presence in the platform of the trusted module when installing.

- When the user tries to run the software, the software executor takes overall control and makes initial checks at the start of the execution. If these checks are satisfied, the software executor allows the software to run. If additional API calls have been incorporated into the software, these are made to the trusted module at various points during runtime.

- At the same time as such checks are made, a record is made in the trusted module if the software were executed successfully. In some models of payment the usage reports could be sent to the clearinghouse or registration body. Payment for a certain number of executions of software could easily be modelled, e.g. using smart cards.

Example B

[0157] The second example uses the trusted module as a generic dongle by encrypting sections of, or all of, the data. Again, there is integrity checking of the data to be executed by the secure loader, and integrity checking of the secure loader, secure executor, software executor and secure transfer code. The trusted module's trusted identity (private crypto key) is used to perform strong authentication. Optionally, applications may be run within a trusted module or smart card.

[0158] The general advantage of such a licensing system is that the flexibility of licence management systems can be combined with the greater degree of hardware security, without the drawbacks of dongles.

[0159] In particular, problems with current licensing systems are countered as follows:

- Bypassing of licensing checks is countered by an integrity check on the platform, which will fail if the trusted device is removed or tampered with or the licensing software is altered.

- A drawback of current generic methods of data protection is that, even if the data is protected up to the point of execution, once the executable is unlocked or made available for use, it can potentially be copied and used freely. Although it will still be possible to copy the data, the data cannot be executed on any other secure client platform that incorporates this invention without a requisite licence.

- The dongle is generic rather than tailored to specific applications.

- There is flexibility in payment and licensing models (including allowing a combination of different types of licensing).

- There is an improvement upon generic dongles such as Wave Systems WaveMeter in that it allows avoidance of universal system keys within the hardware device and allows the secret keys of the developer and of the hardware to remain secret. This is especially important if the third parties are non-trusted, since neither the clearinghouse, nor anyone else, will be able to make use of the protected data, since they will not know the unlock key. This is an improvement on current systems, where this key will be known by the clearinghouse.

- The automated transfer of licences between trusted modules avoids the key management problem.

- Each developer has a choice of either generic or specific content protection - K (or K') can potentially be different for each customer, if desired. This gives the developer greater flexibility and allows him/her to balance effort against extra security. More generally, each type of licensing model (for example, corresponding to examples A, B or C) can be used based on the data shipped to each customer being the same, or individually customised (and hence not usable on other machines). A combination of these methods could be used on the same platform. Therefore, the choice is given to the developer about what type of data protection he would like to

use. The developer just makes the unlock key, or type of generic protection, different for each customer, or the same. The client platform does not have to be informed about this choice.

[0160] In this example:

- A generic secure executor, secure loader and secure key transfer code is included in every trusted computer platform. The code will not be loaded if the integrity check fails, and in this case the complete client platform integrity check should fail, as described previously in this document.

- An end-user A registers his client machine (trusted device ID) with a developer, server or clearinghouse C (according to the payment model) and arranges to make appropriate payment in order to receive some data. As an alternative, the hardware device could be charged up in advance, and the data purchase recorded on this device and reported back to C at a later date.

- As part of the initialisation process, authentication between communicating parties within the licensing system will take place before (or at the same time, by the protocols being incorporated) as exchange of session keys for confidentiality of the messages.

- Authentication: There is authentication from C to the client's tamper-proof device. This is done using a standard protocol incorporating a challenge from A's trusted module to C containing a nonce (to give protection against replay attacks), and C responding with a message containing this nonce, digitally signed using its private code-signing key. Optionally, there is authentication from A's tamper-proof device to C. A public key certificate giving the public key W corresponding to C's private code signing key is transferred to the trusted component of the end-user (in some cases (e.g. upgrades) it will already be present in the trusted module). This is for the machine to be able to check the vendor's identity, and the integrity of the upgrade data it will receive later. If a user-based licensing model is to be used, the transfer will be to the portable trusted device (e.g. smart card). C is also given the public key corresponding to a private key P in A's tamper-proof device. This is needed for some types of authentication of A to C, and when using symmetric encryption keys set up using an asymmetric key pair (see below). In an analogous manner, public key certificates between the developer and the clearinghouse, if these are separate parties, will need to be exchanged initially and appropriate authentication carried out. The same protocols can be used as described above.

- Data encrypted using a symmetric key K is signed under C's private code signing key (e.g. using Microsoft's Authenticode) and sent by C to A's machine to the end-user. K can potentially be different for each customer, if desired. This data is transferred to the end-user by any convenient means (for example, internet or satellite broadcast), since it is the unlock key that needs to be protected. An option is to use instead a private key K', since time taken to encrypt is probably not an issue at this stage.

- Confidentiality: If there is a separate developer and clearinghouse, a protocol is used between the developer and the clearinghouse to set up a symmetric key pair, that can be used to encrypt communication between them, for example about payment and usage of data. By these means neither party knows the other party's secret key. The contents of each message which is to be protected are encrypted using a randomly generated DES key, and with it the symmetric key is transferred RSA-encrypted using the public key of the intended recipient. In this case too, a public key certificate corresponding to the other party will need to be installed in each party initially. If checks for authenticity and integrity are added, the following protocol results for each message: The sender generates a DES key (using a random number generator, and making sure these keys are only used once). The sender then uses it to encrypt the data D, and then encrypts that DES key using the recipient's RSA public key. Then the sender signs a hash of all this information to offer authentication and integrity, and sends the encrypted data and encrypted DES key together with this signature. Note that the sensitive data D is stored encrypted with the DES key. Only the recipient should then have the RSA private key to decrypt the DES encryption key, and use it to decrypt the data D.

- All communications between A and C are encrypted using DES session keys, as discussed in the previous stage.

- In addition, the symmetric unlock key corresponding to K (or, alternatively, the public key corresponding to K') is encrypted using A's public key and signed using C's private code signing key and is sent to the end-user's tamper-proof component in order to allow the data to run.

- Once received by the end-user platform, an integrity check is performed by the secure loader on the data by checking the signature using W and verifying whether it is from the expected source.

- If the integrity check succeeds, the data is installed on the platform and the trusted component records

this event. Otherwise, an error message will be generated and the data will not be loaded.

- The tamper-proof device associated with the end-user's PC is the only one able to make use of this information, and obtain the unlock key. The key transfer code checks the message for integrity and authentication, decrypts the unlock key and stores this on the trusted module, associated with the data.
- When the user wishes to run the data, the secure executor decrypts the data using the unlock key and allows the data to run. The actual functionality of the unlock key could vary: for example, part of the program could be decrypted upon start up or installation, or the key itself could be formed using the identity of the tamper-proof component as input.
- The tamper-proof component keeps a log to monitor usage of the data locally, and in a trusted fashion.

Example C

[0161] The third example is of licensing via consulting database or profile information associated with the identity of the trusted module.

[0162] This involves updating a licence database entry in return for registration and payment. There are two main options using this approach.

Example C1

The first is that the secure executor checks in a database against the trusted module ID entry for an unlock key for the data. The data is protected via encryption or partial encryption using a key, and hence can be freely distributed without fear of piracy.

Example C2

The second is that the secure executor or software executor checks in a database against the trusted module ID entry for permissions for running a piece of data. An entry corresponding to the trusted module's ID is updated to show permission to run a particular application, and the secure executor or software executor will only allow data to run once permissions on this database have been checked. In this case the data will be generic and unprotected, and can be copied freely, but of course not run on this type of platform if the requisite permissions are not in place. The trusted module will update its log if the secure executor has allowed the data to run. In the case of using a software executor to perform the checks, the software executor associated with the application to be run calls the trusted module, the trusted module performs the licence check, and then if this check is successful the software executor passes the call to the OS to run the application.

[0163] The advantages of this approach are:

1) The flexibility of licence management systems can be combined with the greater degree of hardware security, without the drawbacks of dongles.

2) A major motivation for using such a method would be for reasons of key management. In particular, issuing replacement passwords is troublesome. This method gets round this problem, in that it is only a database that has to be updated.

3) If directory systems are already in place, this licensing method would be a natural choice as it would not require much extra investment to provide a secure licensing check.

4) Example C1 above corresponds to another method of giving an unlock key to the client machine, as compared with example B. This could be preferred for two reasons. First, directory systems might be in place and a favoured solution for a particular corporation. Secondly, this method can allow non-permanent storage of unlock keys, allowing floating licences, which example B does not.

[0164] A licensing procedure which could be used at present would be to check fingerprinting information against a licensing database to see whether there was a valid licence corresponding to that fingerprint. The application would be allowed to run or not depending upon this information. However, this method is not really used because:

- The licence-checking code could at present easily be bypassed.
- There is an overhead involved in generating the databases and keeping them up to date.
- It is possible to spoof ID to gain access to information which is licensed to another machine or user.

[0165] However, via using a tamper-proof device in conjunction with integrity checking of the associated licence-checking code, an analogous method can be used.

[0166] The method overcomes the problems associated with the existing procedure:

- Directory structures can be extended to allow licensing (cf. licence management) - these structures are already there, and allow integration with additional functionality. The licence database could be in the form of local records stored in the trusted

component, a record stored in a server (and consulted or stored locally when needed), or a centrally-maintained directory service, where appropriate information about access is stored. Indeed, a combination of these could be used. Directory standards, commonly known as X.500, provide the foundations for a multi-purpose distributed directory service that interconnects computer systems belonging to service providers, governments, and private organisations. It would be straightforward to modify such directories so that for computer network users, a look-up of a person's user ID or machine ID could return information including details of the applications licensed to that individual or machine, respectively.

- There is an integrity check on licence-checking code, and also on the data. Associated software on the computer platform would check if the user or machine had permission to run the application, and allow or disallow this as appropriate. Alternatively, if the data was protected, say by encryption, different data access keys could be stored in the directory, and access to them obtained in this manner, via the associated software.

- Better authentication allows a directory/profile approach. Trusted ID within the trusted module (possibly combined with biometrics, if it is user ID) allows stronger authentication and helps prevent spoofing. (A more trustworthy machine or user identity makes this method less open to abuse, for example by another user's identity being given.) Keys can also be stored more securely. Optionally, software could be added to ensure that the system meters data usage, and store this within the tamper-proof device. If a smart card were used, the check in the profile would be against the user ID, single sign on would mean that the card would not have to be left within the reader, and location independence would also be gained.

[0167] With reference to the two main options of licensing using the method C given above, let us consider the first case initially, C1:

- The secure executor is generic and is integrated with the platform in order to stop theft of the unlock key. This is possible because the same procedure is used with different data, and only the data name and associated key will differ in each case. The secure executor and secure loader are stored together with a hashed version signed with the manufacturer's private key. The manufacturer's public key certificate will be included in every platform. Upon boot/installation of the platform, the package is verified by hashing, and comparison with the decrypted signature to check integrity, using the public key certificate. The code will not be loaded if the integrity check fails, and in this case the complete platform integrity fails.

- Upon registration of the trusted module ID and payment, the clearinghouse or developer causes the unlock key of the data K to be inserted into the database entry corresponding to the trusted module ID (this may actually be carried out by a third party, with authorisation from the clearinghouse or developer).

- The public key certificate for C is installed by C into the client trusted module. A suitable protocol which would incorporate authentication from C to the trusted module would be that, in response to a request for authentication from the trusted module incorporating a nonce generated by the trusted module, C returns a message which includes its public key certificate and the nonce, signed with its private key. The trusted module can then check that the message came from C.

- The software or other data to be protected is encrypted using a symmetric key corresponding to K and signed under C's private code signing key (e.g. using Microsoft's Authenticode) and sent by C to A's machine to the end-user. K can potentially be different for each customer, if desired. This data can be transferred to the end-user by any convenient means (for example, internet or satellite broadcast), since it is the unlock key that needs to be protected.

- Once received by the end-user platform, an integrity check is performed by the secure loader on the data by checking the signature using the public key corresponding to C's private code signing key.

- If the integrity check succeeds, the software or other data is installed on the platform and the trusted component records this event. Otherwise, an error message will be generated and the data will not be loaded.

- When the user wishes to run the data, the secure executor
  - checks the trusted module ID, for example by authentication involving a nonce to counter replay attacks and signed communication
  - checks the database entry of the trusted module ID and retrieves the unlock key K
  - allows the data to run, or not, as appropriate.

- The tamper-proof device then updates its logs to record if the data has been run. If a user has logged

in with a smart card, the userID of this device can be noted, along with the data and time

[0168] A variation is to store the unlock key within the trusted module, once it has been retrieved, along with the data name, so that the database lookup procedure need not be carried out again for this particular data. Future requests for running the data would result in a challenge from the software executor to authenticate the trusted module ID, check the unlock key, use this to decrypt the data and allow the data to run (in the same manner as in example B above)

[0169] Now moving on to consider the second case, C2, when the secure licence permissions for running a piece of data are checked for. There are two possible sub-models, depending upon whether the secure executor (a generic piece of code that is incorporated into the platform) communicates with the operating system and initiates the data execution process, or whether a (customised) software executor, shipped together with each piece of data from the clearinghouse or developer, communicates with the operating system and initiates the process. Within each, there is a choice about whether to load licensing information into the trusted module itself, or refer to an external database.

[0170] The data itself is not protected in this model. If greater confidentiality of the data is required, variants of examples A or B should be used instead.

[0171] Considering the first generic sub-model, this is very similar to that described in the key checking case of example C1.

- A public key certificate corresponding to the party running the database is installed at the clearinghouse or developer, and vice versa.
- Upon registration and/or payment for the data by the end-user, the clearinghouse or developer C (depending on the payment model) is told the trusted module ID.
- A public key certificate corresponding to the client's trusted module is installed at the clearinghouse or developer (if not already present), and vice versa. A suitable protocol which would incorporate authentication from C to the trusted module would be that, in response to a request for authentication from the trusted module incorporating a nonce generated by the trusted module, C returns a message which includes its public key certificate and the nonce, signed with its private key. The trusted module can then check that the message came from C. An analogous protocol would be used for public key certificate transfer and authentication from the trusted module to C.
- C sends the application or other data which is to be protected to the client, in the following manner: The data is signed by using a hashed version of the message signed by the sender's private key appended to the message, so that the receiver can check the integrity of the message. Explicitly, the developer hashes M, which is the data together with any associated software executor, and signs it with his private key (Sprk) to produce a signature $\Sigma_{Sprk}(h(M))$. Then he sends this signature together with M.
- The secure loader will then check the signature, using the developer's public key, and therefore retrieve the message hash. This guarantees that the developer is the one whose public key has been used to check the signature. Having the message, and the message hash, the secure loader, via the trusted module, can then compute the hash of the message and compare it to the message hash it has decrypted. This checks for integrity of the code. Furthermore the integrity checking mechanism should prevent replay-attacks by some standard mechanism - such as using a nonce. If the integrity check works, the secure loader installs the data. This ensures that modified data (e.g. without API calls) cannot be run, viruses are not introduced, etc.
- C authorises the database entry corresponding to the trusted module ID to be updated, according to the data purchased. The party running the database communicates with the clearinghouse or developer using public key cryptography setting up shared symmetric keys, and by each signing their messages. The contents of each message that is to be protected are encrypted using a randomly generated DES key, and transferred together with the symmetric key which is RSA-encrypted using the public key of the intended recipient. If checks for authenticity and integrity are added, the following protocol results for each message:
- The sender generates a DES key (using a random number generator, and making sure these keys are only used once). The sender then uses it to encrypt the data D, and then encrypt that DES key using the recipient's RSA public key. Then the sender signs a hash of all this information to offer authentication and integrity, and sends everything together with this signature. Only the recipient should then have the RSA private key to decrypt the DES encryption key, and use it to decrypt the data D.
- Upon a request to run a piece of data from the user, the secure executor consults the database containing licensing information to see whether permission to run the data is associated with the trusted module ID of the current platform. If it is not, an error message will be generated to the user and the data will not be allowed to run. If it is, the secure executor will ask the OS to run the data.

[0172] Considering now the second sub-model, one instantiation of the model of having a specific software executor per application would be as follows.

- Upon registration and/or payment for the data, the clearinghouse or developer C (according to the exact payment model) authorises the database entry corresponding to the trusted module ID to be updated, according to the data purchased. (Prior to this, public key certificates between these bodies will have been exchanged: a suitable protocol which would incorporate authentication from C to the trusted module would be that, in response to a request for authentication from the trusted module incorporating a nonce generated by the trusted module, C returns a message which includes its public key certificate and the nonce, signed with its private key. An analogous protocol would be used for public key certificate transfer and authentication from the trusted module to C.) The party running the database communicates with the clearinghouse or developer using public key cryptography setting up shared symmetric keys, and by each signing their messages

- The clearinghouse or developer sends the data, associated with a (customised) software executor, to the client. The software executor is customised such that the public key of the trusted module is inserted into the software executor (alternatively, a shared key is set up between the secure executor and the trusted module). Both the data and the software executor are hashed and signed with the clearinghouse/developer's private key, and the public key corresponding to this is stored on the trusted module.

- The secure loader integrity checks the data and the software executor: upon installation, the package is verified by hashing and comparison with the decrypted signature (using the public key in the trusted module).

- The data and software executor are not loaded if the digital signature does not match what is expected

- When the user wishes to execute the data, the OS sends a message to the software executor corresponding to that data. The software executor then issues a challenge/response to the secure executor, by means of sending a random number (nonce), together with the application's title. In addition, a smart card ID is sent, if that was used to log in to the client machine and hot-desking is the licensing model to be used.

- The secure executor
    - checks to see whether the data is licensed to run on the trusted module machine ID in the profile stored within the trusted module, or
    - checks to see whether the data is licensed to run according to the user ID of a smart card which has been inserted in the profile stored within the trusted module, or
    - consults, or downloads part of an external database to form a profile within the trusted module, to see whether the application is licensed in the manner described above.

- If there is no valid licence, the secure executor returns an error message, from which the software executor can determine the exact type of problem with licensing and notify the OS appropriately. If there is a valid licence, the secure executor returns a message incorporating the nonce and data reference, signed and encrypted using the trusted module's private key

- The software executor verifies if the secure executor's reply is correct using the trusted module's public key, and either passes the call to the OS to execute the data or sends an error message to the OS as appropriate.

<u>Example D</u>

[0173] The fourth example is of using the trusted module as a dongle by fingerprinting the trusted module.

[0174] This differs from current fingerprinting techniques in that it uses a trusted identity within the hardware (viz. the non-secret trusted module identity), integrity checking of the application to be run, integrity checking of associated application-enabling software and uses secure audit within the hardware. Optionally, an unlock key can be generated within the software executor on the client machine, rather than remotely. The trusted module will have to contact the vendor in order to obtain a key, the protected data, and the associated software executor, which will enable the decryption key to be generated locally using the trusted module ID. The data could be generically encrypted and shipped, because a single key could be used to decrypt it, or different keys could be used for each end-user (which is more secure).

[0175] This method is a variant of B, and provides an alternative to the approach used in B. It differs in that:

- The unlock key can be generated within the software executor or secure executor on the client machine rather than remotely

- The key transferred from the clearinghouse to the client machine is not the unlock key, but a key from which this can be derived using an algorithm found

in the software executor, and fingerprinting details of the trusted module. It would be better to use the software executor than the secure executor, since the techniques used to derive the unlock key can vary between developers.

[0176] The flexibility of licence management systems can be combined with the greater degree of hardware security, without the drawbacks of dongles. This method counters problems associated with current methods of licence protection including the following:

- Attacks using machines pretending to be other machines. The machine ID, which is the device ID for internal components, is trustworthy. This is useful for licensing for more secure logging, allowing greater licensing information and models, and authentication. PC fingerprints are less easy to fake than at present because device ID is more reliable than what is used at present for PC fingerprinting, i.e. hard disk ID, BIOS serial number, network ID card, etc. Such reliable identification helps against attacks using machines pretending to be other machines.

- Data can be bypassed or altered, and so software-only protection is subject to a universal break. The actions taken to perform the security, fingerprinting and authentication need to be hidden from a hacker. However, because all information is stored on the PC and functions are done using the PC's processor, these actions can be traced by a debugger. The only way to safeguard these actions from a debugger is to use operating system or machine specific exceptions, like Ring Zero in Windows. While this improves security by blocking most debuggers, it does not stop chip simulators which are widely available for PC processors like Intel's Pentium. In addition, this makes the software only solution machine specific and requires a version for each of the various platforms. Many software only protection suppliers are small and cannot provide timely protection modules for all the various combinations of applications and operating environments. This leads to incompatibilities that irritate the user and cost the developer support time. Since the same authentication action must be performed on only a few identifiable PC components before any program is loaded, the hacker has relatively little code to trace; therefore, once the loading sequence is understood, the protection for all applications using the software only scheme can be easily broken. Integrity checks on the platform and software allow integrity checks on associated licensing-checking and uploading software and avoid data being bypassed or altered. The licensing aspects described are not reliant on the PC processor - the algorithm function is performed within the trusted hardware, where no debugger or chip simulator can expose the process.

- A single LMF can manage all features of all of the applications sold by one developer. But there needs to be a separate arrangement with each developer, and possibly clashes between the different licence managers. It would be better to have just one licence manager per user site, and each developer connect into this. This model is even more general, and could cover all developers.

- Software solutions give slow encryption, are less secure and can only provide a limited amount of security to stored data. Slow encryption is of limited use and makes using encryption in bulk for all communications impractical. End users can either wait longer for their communication and applications, or choose to encrypt only small pieces of the communication. Hardware encryption is faster. By using fast encryption for all communication, it can be transparent - a better solution than partial encryption. Hardware is widely recognised as being more secure because it can be encased in a tamper resistant package, and its interface can be more securely controlled. Hardware solutions allow much greater protection of sensitive data such as keys and user information.

[0177] There are two main types of use of example D.

- First, in situations where a machine-based licensing model is most appropriate:
  - Data S is encrypted using a key K.
  - A user registers with the clearinghouse/developer C, there is mutual authentication and C is given the trusted module ID.
  - C sends the encrypted data plus associated software executor to the user by any convenient means, signed and hashed.
  - The secure loader on the client computer checks integrity and installs the data S if the integrity check succeeds.
  - Symmetric cryptography is used to transfer the unlock key from C to the trusted module. This key will not be useful to another machine, and therefore does not need to be protected from third parties as much as in Example B, when the key transferred could be a system-level unlock key.
  - The software executor calculates the decryption key corresponding to K from the unlock key

and the trusted module ID, using an algorithm pre-stored within it by C or a third party trusted by C.

- The decryption key is used to decrypt the data and allow it to run.

- Secondly, in situations where a user-based licensing model is required

  - Data S is encrypted using a key K.

  - A user registers with the clearinghouse/developer C, there is mutual authentication and C is given the smart card ID.

  - C sends the encrypted data plus associated software executor to the user by any convenient means, signed and hashed.

  - The secure loader on the client computer(s) selected by the user checks integrity and installs the data S if the integrity check succeeds

  - The unlock key is transferred by any convenient means from C to the user. This key is not particularly confidential, and can be transferred by telephone or electronically.

  - The user logs in to a trusted platform computer and inserts the smart card in the reader

  - When the user tries to run the data, he is prompted to type in the unlock key

  - The software executor calculates the decryption key corresponding to K from the unlock key and the smart card ID, using an algorithm pre-stored within it by C or a third party trusted by C.

  - The decryption key is used to decrypt the data and allow it to run.

Example E

[0178] There is an option to use any of the examples A-D above, but running applications suitably segmented within a trusted module: as well as running applications on the platform in a similar manner to current practice, there are additional options to run the applications within the internal machine trusted module, within a portable trusted module such as a smart card, or using a combination of any of these. State-of-the-art techniques known to an expert in the field which have been patented for running multiple applications on a smart card would be used.

Example F

[0179] The final example is of how a combination of multiple trusted devices can be used to licence data in a flexible manner. The combination of an internal machine trusted module and a portable trusted module such as a smart card is considered, for the particular case in which the hot-desking licensing model is used, and the OS communicates with the software executors. An analogous procedure would be used for the model described in Figure 19.

- Upon registration and/or payment for the data, the clearinghouse or developer (according to the exact payment model) authorises the database entry corresponding to the trusted module ID to be updated, according to the data purchased. (Prior to this, there will be mutual authentication, as described in previous examples, and public key certificates between these bodies will have been exchanged). The party running the database communicates with the clearinghouse or developer using public key cryptography setting up shared symmetric keys, and by each signing their messages. The contents of the message which is to be protected are encrypted using a randomly generated DES key, and transferred together with the symmetric key which is RSA-encrypted using the public key of the intended recipient, according to a standard protocol.

- The clearinghouse or developer sends the data, associated with a (customised) software executor, to the client. The software executor is customised such that the public key of the trusted module is inserted into the software executor (alternatively, a shared key is set up between the secure executor and the trusted module). Both the data and the software executor are hashed and signed with the clearinghouse/developer's private key, and the public key corresponding to this is stored on the trusted module.

- The secure loader integrity checks the data and the software executor: upon installation, the package is verified by hashing and comparison with the decrypted signature (using the public key in the trusted module)

- The software executor is not loaded if the digital signature does not match what is expected.

- Upon sign-on using the smart card, public key certificates of the smart card and trusted module are exchanged for future communication (if this has not already been done), and there is mutual authentication between the trusted module and the smart card.

- The trusted module stores the (current) smart card ID.

- When the user wishes to execute some data, the software executor corresponding to that data issues a challenge/response to the secure executor, by means of sending a random number (nonce), together with a reference to the data.

- The secure executor makes an appropriate licensing check on the data, using the smart card ID, or else by obtaining some information stored on the smart card. For example, using the licensing model described above, the secure executor:

  - checks whether the data is licensed to run according to the user ID of the smart card which has been inserted, in the profile stored within the trusted module, or

  - checks whether the data is licensed to run on the trusted module ID in the profile stored within the trusted module, or

  - consults or downloads part of an external database to form a profile within the trusted module to see whether the data is licensed in the manner described above.

- If there is no valid licence, the secure executor returns an error message, from which the software executor can determine the exact type of problem with licensing and notify the OS appropriately. If there is a valid licence, the secure executor returns a message incorporating the nonce and data reference, signed and encrypted using the trusted module's private key.

- The software executor verifies if the secure executor's reply is correct using the trusted module's public key, and either passes the call to the OS to execute the data or sends an error message to the OS as appropriate.

- The log is held within the machine trusted module rather than the smart card, and is updated appropriately.

[0180] It should be noted that the embodiment of the invention has been described above purely by way of example and that many modifications and developments may be made thereto within the scope of the present invention.

Claims

1. A computer platform having

   a trusted module which is resistant to internal tampering and which stores a third party's public key certificate;
   means storing licence-related code comprising at least one of:

   a secure executor for checking whether the platform or a user thereof is licensed to use particular data and for providing an interface for using the data and/or for monitoring its usage; and
   a secure loader for checking whether the platform or a user thereof is licensed to install particular data and/or for checking for data integrity before installation; and

   means storing a hashed version of the licence-related code signed with the third party's private key;
   wherein the computer platform is programmed so that, upon booting of the platform:

   the licence-related code is integrity checked with reference to the signed version and the public key certificate; and
   if the integrity check fails, the licence-related code is prevented from being loaded.

2. A computer platform as claimed in claim 1, wherein the integrity checking is performed by:

   reading and hashing the licence-related code to produce a first hash;
   reading and decrypting the signed version using the public key certificate to produce a second hash; and
   comparing the first and second hashes.

3. A computer platform as claimed in claim 1 or 2, wherein the licence-related code also includes secure key-transfer code for enabling a licence key to be transferred between the trusted module and a further trusted module of another computer platform.

4. A computer platform as claimed in any preceding claim, wherein the licence-related code also includes a library of interface subroutines which can be called in order to communicate with the trusted module.

5. A computer platform as claimed in any preceding claim, wherein the licence-related code includes, for at least one group of data, a (or a respective) software executor which specifies the respective group of data and which is operable to act as an interface to that group of data.

6. A computer platform as claimed in any preceding claim, wherein the means storing the licence-related code and/or the means storing the hashed version of the licence-related code are provided, at least in part, by the trusted module.

7. A computer platform as claimed in any preceding claim, wherein the trusted module and an operating system of the platform have a dedicated communications path therebetween which is inaccessible to other parts of the computer platform.

8. A computer platform as claimed in any preceding claim, wherein:

    the operating system is operable to request the secure loader to licence-check whether the platform or a user thereof is licensed to install that particular data and/or to check the integrity of that data;
    in response to such a request, the secure loader is operable to perform such a check and respond to the operating system with the result of the check; and
    in dependence upon the response, the operating system is operable to install or not to install the particular data.

9. A computer platform as claimed in claim 8, wherein the operating system is programmed to install the particular data only in response to the secure loader.

10. A computer platform as claimed in claim 8 or 9, wherein:

    the trusted module stores a public key certificate for a party associated with the particular data to be installed;
    the operating system is operable to include, in the request to check, the particular data together with a hashed version thereof signed with a private key of the associated party;
    in performing the check, the secure loader is operable:

        to hash the particular data included in the request to produce a third hash;
        to decrypt the signed hashed version in the request using the public key certificate for the associated party to produce a fourth hash; and
        to generate the response in dependence upon whether or not the third and fourth hashes match.

11. A computer platform as claimed in claim 10 when dependent directly or indirectly on claim 5, wherein the request to check includes the software executor for the particular data.

12. A computer platform as claimed in claim 8 when dependent on claim 5, or any of claims 7 to 11 when dependent thereon, wherein:

    the software executor (or at least one of the software executors) is operable to request the trusted module to install particular data;
    in response to such a request, the secure loader within the trusted module is operable to licence-check whether the platform or a user thereof is licensed to install that particular data and/or to check the integrity of that data and to respond to the operating system with the result of the check; and
    in dependence upon the response, the operating system is operable to install or not to install the particular data.

13. A computer platform as claimed in claim 12, wherein the operating system is programmed to install the particular data only in response to the trusted module.

14. A computer platform as claimed in claim 12 or 13 when dependent on claim 7, wherein the response from the trusted module to the operating system is supplied via the dedicated communications path.

15. A computer platform as claimed in any of claims 8 to 14, wherein, if the check succeeds, the trusted module is operable to generate a log for auditing the particular data.

16. A computer platform as claimed in any of claims 8 to 15, wherein, if the check succeeds, the secure loader is operable to perform a virus check on the particular data.

17. A computer platform as claimed in any of claims 8 to 16, wherein, upon installation, the particular data is installed into the trusted module.

18. A computer platform as claimed in any of claims 8 to 16:

    further including a further, removable, trusted module;
    wherein the platform is operable to perform an authentication check between the first-mentioned trusted module and the removable trusted module; and
    wherein, upon installation, the particular data is installed into the further trusted module.

19. A computer platform as claimed in claim 5, or any

of claims 6 to 18 when directly or indirectly dependent thereon, wherein:

the software executor (or at least one of the software executors) contains a public key of the trusted module and a licensing model for the respective data,
the operating system is operable to request that software executor that its respective data be used;
in response to such a request, that software executor is operable to request the secure executor to licence-check, using its licensing model, whether the platform or a user thereof is licensed to use that data,
in response to such latter request, the secure executor is operable to perform the requested licence-check, to sign the result of the licence check using a private key of the trusted module, and to respond to that software executor with the signed result;
in response to such a response, that software executor is operable:

to check the integrity of the signed result using the public key of the trusted module; and
upon a successful integrity check of a successful licence-check result, to request the operating system to use that data

20. A computer platform as claimed in claim 5, or any of claims 6 to 19 when directly or indirectly dependent thereon, wherein

the software executor (or at least one of the software executors) contains a public key of the trusted module and a licensing model for the respective data;
the operating system is operable to request the secure executor that particular data be used;
in response to such a request, the secure executor is operable to send to the respective software executor a request, signed using a private key of the trusted module, for a licensing model for the particular data;
in response to such latter request, that software executor is operable:

to check the integrity of the request using the public key of the trusted module; and
upon a successful integrity check, to send the licensing model to the secure executor; and

upon receipt of the licensing model, the secure executor is operable:

to perform a licence-check using that licensing model; and
upon a successful licence-check, to request the operating system to use that data.

21. A computer platform as claimed in any preceding claim, wherein:

the secure executor contains at least one licensing model;
the operating system is operable to request the secure executor that particular data be used; and
in response to such a request, the secure executor is operable:

to perform a licence-check using the, or one of the, licensing models; and
upon a successful licence-check, to request the operating system to use that data.

22. A computer platform as claimed in any of claims 19 to 21, wherein the operating system is programmed to use the particular data only in response to the secure executor or the software executor.

23. A computer platform as claimed in claim 6 when dependent on claim 5, or any of claims 7 to 22 when dependent thereon, wherein:

the secure executor contains at least one licensing model;
the software executor (or at least one of the software executors) is operable to request the trusted module that its respective data be used;
in response to such a request, the secure executor within the trusted module is operable:

to perform a licence-check using the, or one of the, licensing models; and
upon a successful licence-check, to request the operating system to use that data

24. A computer platform as claimed in claim 23, wherein the operating system is programmed to use the particular data only in response to the trusted module.

25. A computer platform as claimed in any of claims 20 to 24 when dependent directly or indirectly on claim 7, wherein the request from the secure executor to the operating system to use the data is supplied via the dedicated communications path

26. A computer platform as claimed in any of claims 19

to 25, wherein the trusted module is operable to log the request to the operating system to use the data.

27. A computer platform as claimed in any of claims 19 to 26:

further including a further, removable, trusted module containing a user identity;
wherein the platform is operable to perform an authentication check between the first-mentioned trusted module and the removable trusted module; and
wherein, upon licence-checking, the secure executor or software executor is operable to perform the licence-check with reference to the user identity.

28. A method of transferring a licence (or a key therefor) for data from a first computer platform, as claimed in claim 3 or any of claims 4 to 27 when dependent thereon, to a second computer platform, as claimed in claim 3 or any of claims 4 to 27 when dependent thereon, the method comprising the steps of:

setting up secure communication between the trusted modules,
sending the licence or the key therefor from the first trusted module to the second trusted module using the secure communication, and
deleting the licence or the key therefor from the first trusted module.

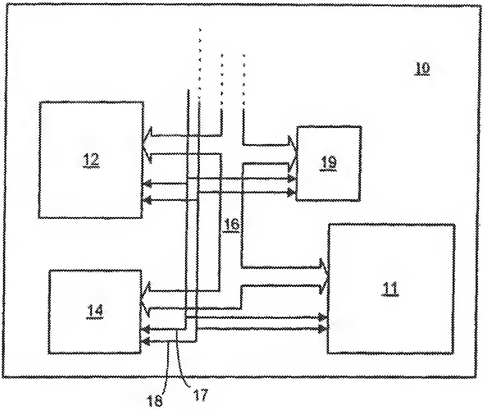


FIGURE 1

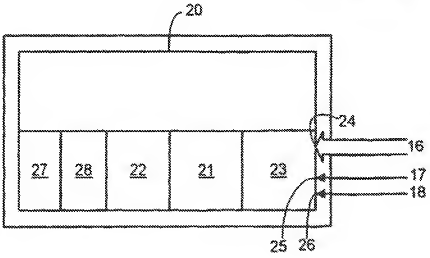


FIGURE 2

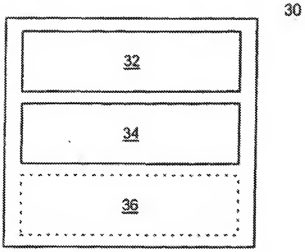


FIGURE 3

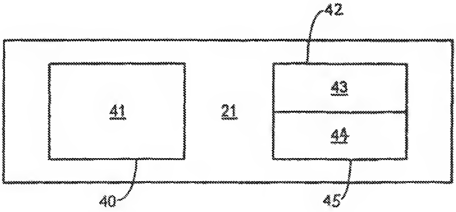


FIGURE 4

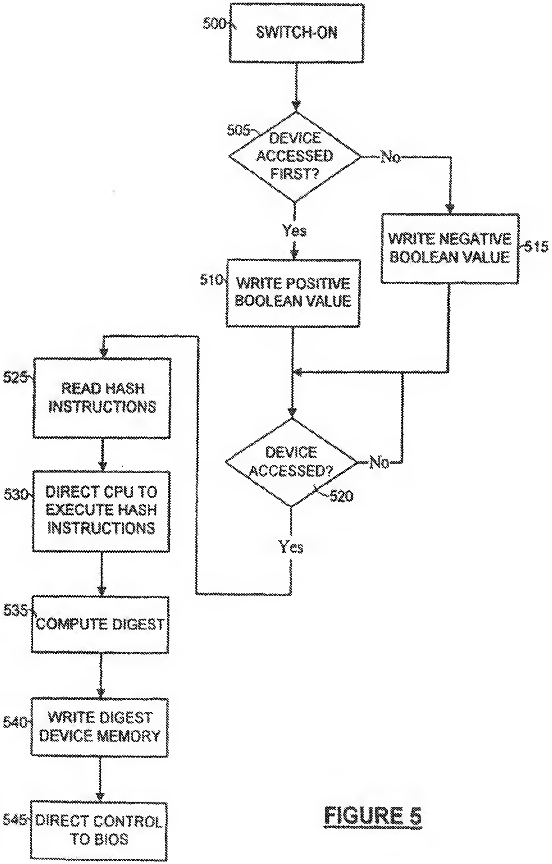

500
SWITCH-ON
505
DEVICE ACCESSED FIRST?
No
Yes
510
WRITE POSITIVE BOOLEAN VALUE
WRITE NEGATIVE BOOLEAN VALUE
515
DEVICE ACCESSED?
No
520
Yes
525
READ HASH INSTRUCTIONS
530
DIRECT CPU TO EXECUTE HASH INSTRUCTIONS
535
COMPUTE DIGEST
540
WRITE DIGEST DEVICE MEMORY
545
DIRECT CONTROL TO BIOS

FIGURE 5

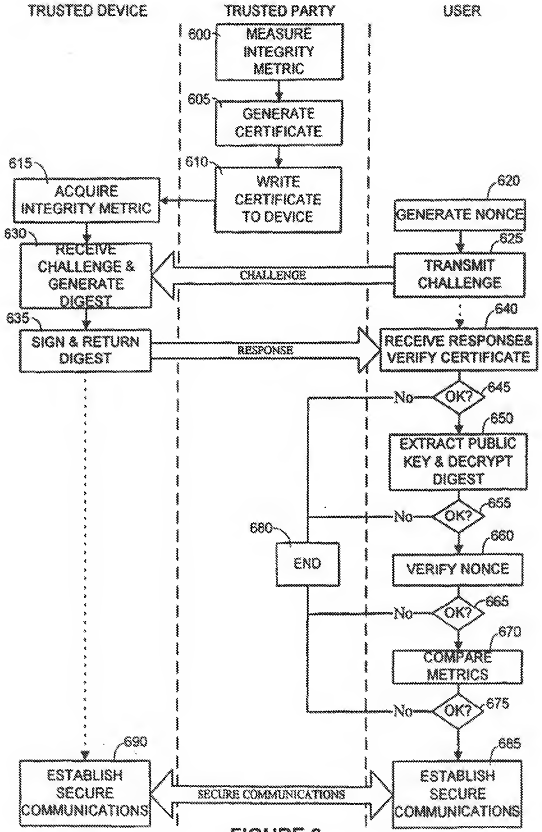

TRUSTED DEVICE
TRUSTED PARTY
USER
600
MEASURE INTEGRITY METRIC
605
GENERATE CERTIFICATE
610
WRITE CERTIFICATE TO DEVICE
615
ACQUIRE INTEGRITY METRIC
620
GENERATE NONCE
625
TRANSMIT CHALLENGE
630
RECEIVE CHALLENGE & GENERATE DIGEST
CHALLENGE
635
SIGN & RETURN DIGEST
RESPONSE
640
RECEIVE RESPONSE& VERIFY CERTIFICATE
645
OK?
No
650
EXTRACT PUBLIC KEY & DECRYPT DIGEST
655
OK?
No
680
END
660
VERIFY NONCE
665
OK?
No
670
COMPARE METRICS
675
OK?
No
690
ESTABLISH SECURE COMMUNICATIONS
SECURE COMMUNICATIONS
685
ESTABLISH SECURE COMMUNICATIONS

FIGURE 6

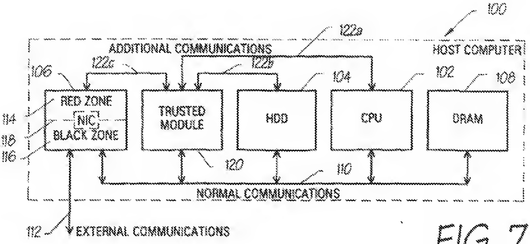

100
122a
ADDITIONAL COMMUNICATIONS
HOST COMPUTER
106
122c
122b
104
102
108
RED ZONE
NIC
BLACK ZONE
TRUSTED MODULE
HDD
CPU
DRAM
114
118
116
120
110
NORMAL COMMUNICATIONS
112
EXTERNAL COMMUNICATIONS

FIG. 7

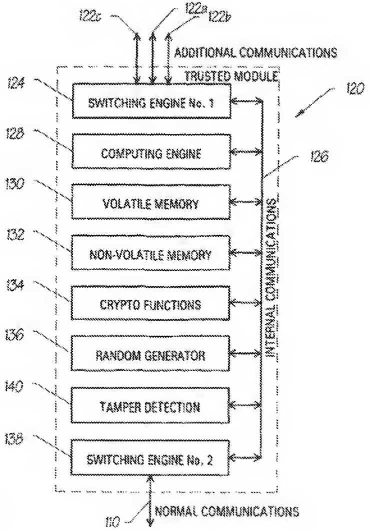

122c
122a
122b
ADDITIONAL COMMUNICATIONS
TRUSTED MODULE
124
SWITCHING ENGINE No. 1
120
128
COMPUTING ENGINE
126
130
VOLATILE MEMORY
132
NON-VOLATILE MEMORY
134
CRYPTO FUNCTIONS
136
RANDOM GENERATOR
140
TAMPER DETECTION
138
SWITCHING ENGINE No. 2
INTERNAL COMMUNICATIONS
110
NORMAL COMMUNICATIONS

FIG. 8

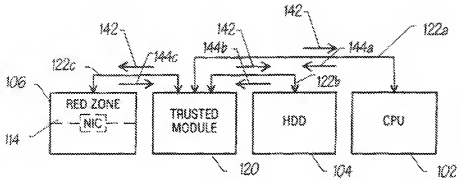


FIG. 9

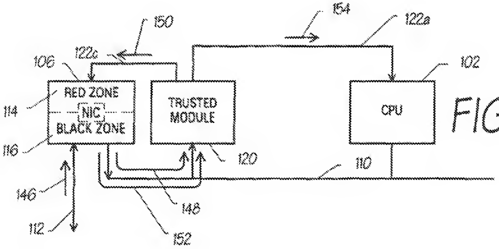


FIG. 10

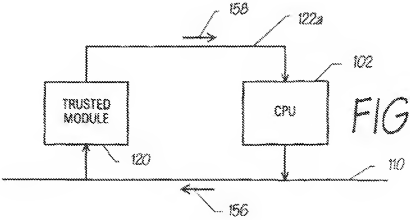


FIG. 11

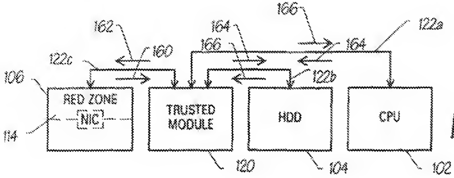

166
162
164
122a
160
166
164
122c
122b
106
RED ZONE
NIC
TRUSTED MODULE
HDD
CPU
114
120
104
102

FIG. 12

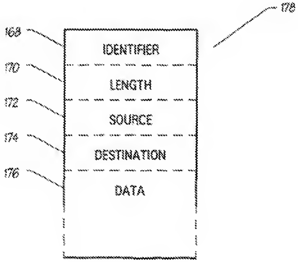

178
168
IDENTIFIER
170
LENGTH
172
SOURCE
174
DESTINATION
176
DATA

FIG. 13

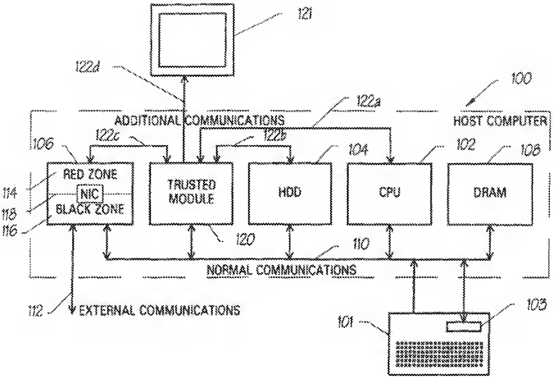

121
122d
100
122a
ADDITIONAL COMMUNICATIONS
HOST COMPUTER
106
122c
122b
104
102
108
RED ZONE
NIC
BLACK ZONE
TRUSTED MODULE
HDD
CPU
DRAM
114
118
116
120
110
NORMAL COMMUNICATIONS
112
EXTERNAL COMMUNICATIONS
101
103

FIG. 14

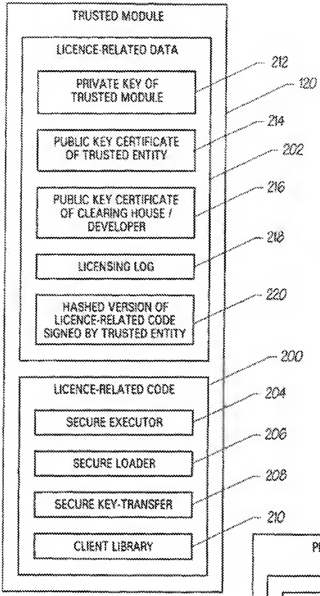

TRUSTED MODULE
LICENCE-RELATED DATA
PRIVATE KEY OF TRUSTED MODULE
PUBLIC KEY CERTIFICATE OF TRUSTED ENTITY
PUBLIC KEY CERTIFICATE OF CLEARING HOUSE / DEVELOPER
LICENSING LOG
HASHED VERSION OF LICENCE-RELATED CODE SIGNED BY TRUSTED ENTITY
LICENCE-RELATED CODE
SECURE EXECUTOR
SECURE LOADER
SECURE KEY-TRANSFER
CLIENT LIBRARY
212
120
214
202
216
218
220
200
204
206
208
210

FIG. 15

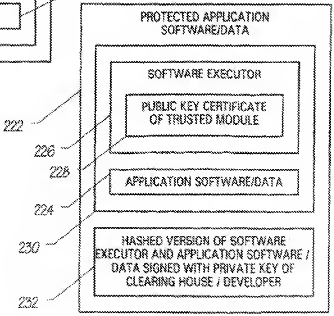

PROTECTED APPLICATION SOFTWARE/DATA
SOFTWARE EXECUTOR
PUBLIC KEY CERTIFICATE OF TRUSTED MODULE
APPLICATION SOFTWARE/DATA
HASHED VERSION OF SOFTWARE EXECUTOR AND APPLICATION SOFTWARE / DATA SIGNED WITH PRIVATE KEY OF CLEARING HOUSE / DEVELOPER
222
226
228
224
230
232

FIG. 16

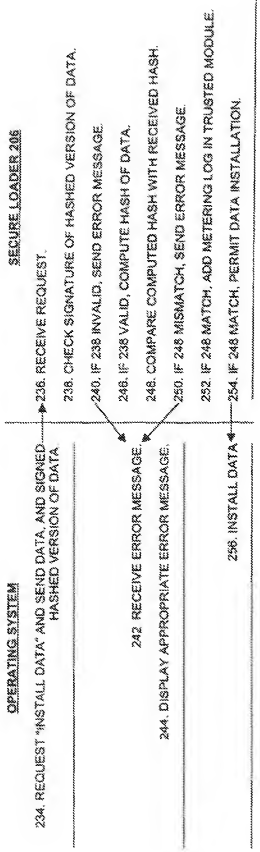

OPERATING SYSTEM
SECURE LOADER 206
234. REQUEST "INSTALL DATA" AND SEND DATA, AND SIGNED HASHED VERSION OF DATA
236. RECEIVE REQUEST.
238. CHECK SIGNATURE OF HASHED VERSION OF DATA.
240. IF 238 INVALID, SEND ERROR MESSAGE.
242. RECEIVE ERROR MESSAGE.
244. DISPLAY APPROPRIATE ERROR MESSAGE.
246. IF 238 VALID, COMPUTE HASH OF DATA.
248. COMPARE COMPUTED HASH WITH RECEIVED HASH.
250. IF 248 MISMATCH, SEND ERROR MESSAGE.
252. IF 248 MATCH, ADD METERING LOG IN TRUSTED MODULE.
254. IF 248 MATCH, PERMIT DATA INSTALLATION.
256. INSTALL DATA

FIG. 17

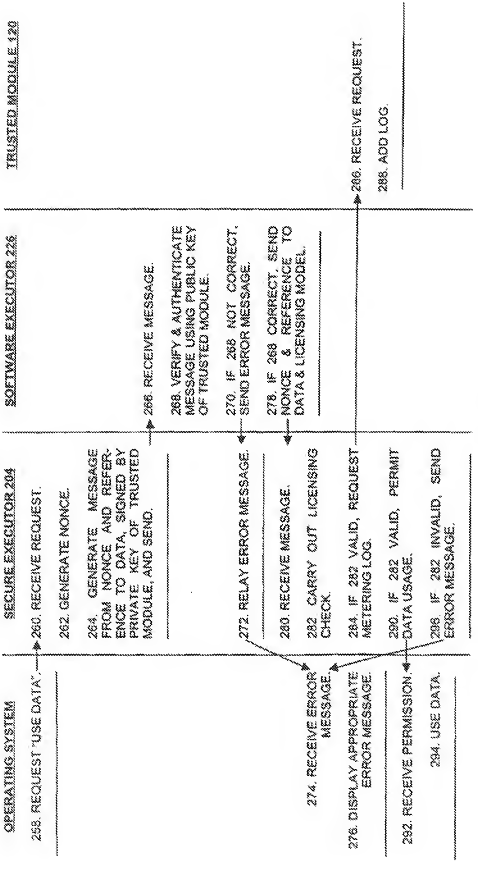

OPERATING SYSTEM
258. REQUEST "USE DATA".
274. RECEIVE ERROR MESSAGE.
276. DISPLAY APPROPRIATE ERROR MESSAGE.
292. RECEIVE PERMISSION.
294. USE DATA.
SECURE EXECUTOR 204
260. RECEIVE REQUEST.
262. GENERATE NONCE.
264. GENERATE MESSAGE FROM NONCE AND REFERENCE TO DATA, SIGNED BY PRIVATE KEY OF TRUSTED MODULE, AND SEND.
272. RELAY ERROR MESSAGE.
280. RECEIVE MESSAGE.
282. CARRY OUT LICENSING CHECK.
284. IF 282 VALID, REQUEST METERING LOG.
290. IF 282 VALID, PERMIT DATA USAGE.
296. IF 282 INVALID, SEND ERROR MESSAGE.
SOFTWARE EXECUTOR 226
266. RECEIVE MESSAGE.
268. VERIFY & AUTHENTICATE MESSAGE USING PUBLIC KEY OF TRUSTED MODULE.
270. IF 268 NOT CORRECT, SEND ERROR MESSAGE.
278. IF 268 CORRECT, SEND NONCE & REFERENCE TO DATA & LICENSING MODEL.
TRUSTED MODULE 120
286. RECEIVE REQUEST.
288. ADD LOG.

FIG. 18

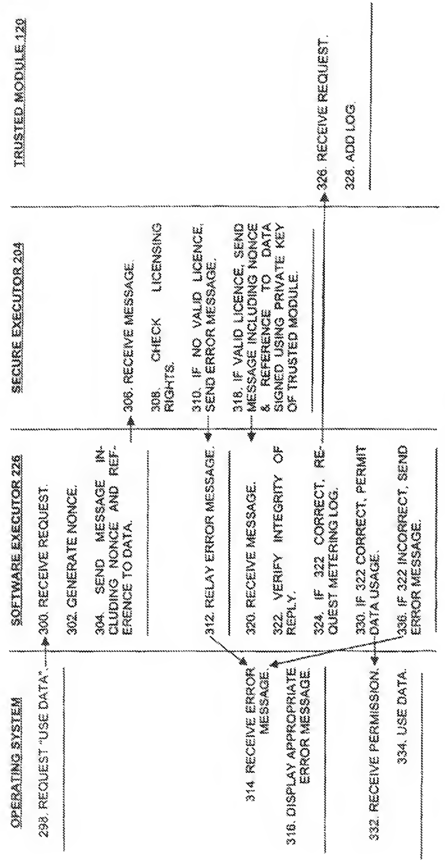

OPERATING SYSTEM
298. REQUEST "USE DATA".
314. RECEIVE ERROR MESSAGE.
316. DISPLAY APPROPRIATE ERROR MESSAGE.
332. RECEIVE PERMISSION.
334. USE DATA.
SOFTWARE EXECUTOR 226
300. RECEIVE REQUEST.
302. GENERATE NONCE.
304. SEND MESSAGE INCLUDING NONCE AND REFERENCE TO DATA.
312. RELAY ERROR MESSAGE.
320. RECEIVE MESSAGE.
322. VERIFY INTEGRITY OF REPLY.
324. IF 322 CORRECT, REQUEST METERING LOG.
330. IF 322 CORRECT, PERMIT DATA USAGE.
336. IF 322 INCORRECT, SEND ERROR MESSAGE.
SECURE EXECUTOR 204
306. RECEIVE MESSAGE.
308. CHECK LICENSING RIGHTS.
310. IF NO VALID LICENCE, SEND ERROR MESSAGE.
318. IF VALID LICENCE, SEND MESSAGE INCLUDING NONCE & REFERENCE TO DATA SIGNED USING PRIVATE KEY OF TRUSTED MODULE.
TRUSTED MODULE 120
326. RECEIVE REQUEST.
328. ADD LOG.

FIG. 19

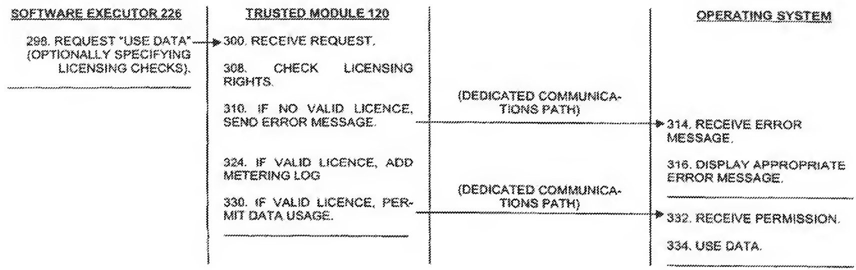

SOFTWARE EXECUTOR 226
TRUSTED MODULE 120
OPERATING SYSTEM
298. REQUEST "USE DATA" (OPTIONALLY SPECIFYING LICENSING CHECKS).
300. RECEIVE REQUEST.
308. CHECK LICENSING RIGHTS.
310. IF NO VALID LICENCE, SEND ERROR MESSAGE.
(DEDICATED COMMUNICATIONS PATH)
314. RECEIVE ERROR MESSAGE.
316. DISPLAY APPROPRIATE ERROR MESSAGE.
324. IF VALID LICENCE, ADD METERING LOG
330. IF VALID LICENCE, PERMIT DATA USAGE.
(DEDICATED COMMUNICATIONS PATH)
332. RECEIVE PERMISSION.
334. USE DATA.

FIG. 20

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 99 30 6415

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.7)
Y	US 5 473 692 A (DAVIS DEREK L) 5 December 1995 (1995-12-05) * the whole document *	1-3,6,28	G06F1/00
A		4,5,7-27	
Y	EP 0 684 538 A (IBM) 29 November 1995 (1995-11-29) * column 5, line 6 - line 53 *	1-3,6,28	
A	US 5 680 547 A (CHANG STEVE MING-JANG) 21 October 1997 (1997-10-21)		
A	WO 98 36517 A (JPC INC) 20 August 1998 (1998-08-20)		
A	EP 0 849 657 A (NCR INT INC) 24 June 1998 (1998-06-24)		

TECHNICAL FIELDS SEARCHED (Int.Cl.7)

G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	28 March 2000	Powell, D

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

ANNEX TO THE EUROPEAN SEARCH REPORT ON EUROPEAN PATENT APPLICATION NO. EP 99 30 6415

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report.
The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

28-03-2000

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5473692 A	05-12-1995	AU 3583295 A	27-03-1996
		EP 0780039 A	25-06-1997
		JP 10507324 T	14-07-1998
		WO 9608092 A	14-03-1996
		US 5568552 A	22-10-1996
EP 0684538 A	29-11-1995	US 5564038 A	08-10-1996
		JP 7319689 A	08-12-1995
		US 5771347 A	23-06-1998
US 5680547 A	21-10-1997	US 5444850 A	22-08-1995
		AU 1042895 A	15-05-1996
		JP 10511783 T	10-11-1998
		WO 9613002 A	02-05-1996
WO 9836517 A	20-08-1998	US 5953502 A	14-09-1999
EP 0849657 A	24-06-1998	JP 10282884 A	23-10-1998

EPO FORM P0459

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 1 085 396 A1

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
21.03.2001 Bulletin 2001/12

(51) Int Cl.[7]: G06F 1/00

(21) Application number: 99307380.8

(22) Date of filing: 17.09.1999

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE
Designated Extension States:
AL LT LV MK RO SI

(71) Applicant: Hewlett-Packard Company
Palo Alto, California 94304-1112 (US)

(72) Inventor: Proudler, Graeme John
Meade Park, Bristol BS12 6XQ (GB)

(74) Representative:
Lawman, Matthew John Mitchell et al
Hewlett-Packard Limited,
IP Section,
Building 3,
Filton Road
Stoke Gifford, Bristol BS34 8QZ (GB)

(54) Operation of trusted state in computing platform

(57) A computing entity comprises a trusted monitoring component having a first processing means and a first memory means, the trusted monitoring component being a self-contained autonomous data processing unit, and a computer platform having a main processing means and a main memory area, along with a plurality of associated physical and logical resources such as peripheral devices including printers, modems, application programs, operating systems and the like. The computer platform is capable of entering a plurality of different states of operation, each state of operation having a different level of security and trustworthiness. Selected ones of the states comprise trusted states in which a user can enter sensitive confidential information with a high degree of certainty that the computer platform has not been compromised by external influences such as viruses, hackers or hostile attacks. To enter a trusted state, references made automatically to the trusted component, and to exit a trusted state reference must be made to the trusted component. On exiting the trusted state, all references to the trusted state are deleted from the computer platform. On entering the trusted state, the state is entered in a reproducible and known manner, having a reproducible and known configuration which is confirmed by the trusted component.

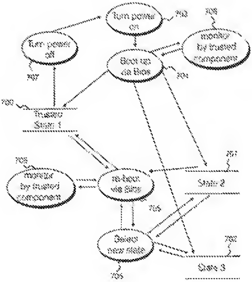


Fig. 7

Description

Field of the Invention

[0001] The present invention relates to the field of computers, and particularly, although not exclusively, to a computing entity which can be placed into a trusted state, and a method of operating the computing entity to achieve the trusted state, and operation of the computing entity when in the trusted state.

Background to the Invention

[0002] Conventional prior art mass market computing platforms include the well-known personal computer (PC) and competing products such as the Apple Macintosh™, and a proliferation of known palm-top and laptop personal computers. Generally, markets for such machines fall into two categories, these being domestic or consumer, and corporate. A general requirement for a computing platform for domestic or consumer use is a relatively high processing power, Internet access features, and multi-media features for handling computer games. For this type of computing platform, the Microsoft Windows® '95 and '98 operating system products and Intel processors dominate the market.

[0003] On the other hand, for business use, there are a plethora of available proprietary computer platform solutions available aimed at organizations ranging from small businesses to multi-national organizations. In many of these applications, a server platform provides centralized data storage, and application functionality for a plurality of client stations. For business use, other key criteria are reliability, networking features, and security features. For such platforms, the Microsoft Windows NT 4.0™ operating system is common, as well as the Unix™ operating system.

[0004] With the increase in commercial activity transacted over the Internet, known as "e-commerce", there has been much interest in the prior art in enabling data transactions between computing platforms over the Internet. However, because of the potential for fraud and manipulation of electronic data, in such proposals, fully automated transactions with distant unknown parties on a wide-spread scale as required for a fully transparent and efficient market place have so far been held back. The fundamental issue is one of trust between interacting computer platforms for the making of such transactions.

[0005] There have been several prior art schemes which are aimed at increasing the security and trustworthiness of computer platforms. Predominantly, these rely upon adding in security features at the application level, that is to say the security features are not inherently imbedded in the kernel of operating systems, and are not built in to the fundamental hardware components of the computing platform. Portable computer devices have already appeared on the market which include a smart card, which contains data specific to a user, which is input into a smart card reader on the computer. Presently, such smart cards are at the level of being add-on extras to conventional personal computers, and in some cases are integrated into a casing of a known computer. Although these prior art schemes go some way to improving the security of computer platforms, the levels of security and trustworthiness gained by prior art schemes may be considered insufficient to enable widespread application of automated transactions between computer platforms. For businesses to expose significant value transactions to electronic commerce on a widespread scale, they require confidence in the trustworthiness of the underlying technology.

[0006] Prior art computing platforms have several problems which stand in the way of increasing their inherent security:

- The operating status of a computer system or platform and the status of the data within the platform or system is dynamic and difficult to predict. It is difficult to determine whether a computer platform is operating correctly because the state of the computer platform and data on the platform is constantly changing and the computer platform itself may be dynamically changing.

- From a security point of view, commercial computer platforms, in particular client platforms, are often deployed in environments which are vulnerable to unauthorized modification. The main areas of vulnerability include modification by software loaded by a user, or via a network connection. Particularly, but not exclusively, conventional computer platforms may be vulnerable to attack by virus programs, with varying degrees of hostility.

- Computer platforms may be upgraded or their capabilities may be extended or restricted by physical modification, i.e. addition or deletion of components such as hard disk drives, peripheral drivers and the like.

[0007] It is known to provide security features for computer systems, which are embedded in operating software. These security features are primarily aimed at providing division of information within a community of users of the system. In the known Microsoft Windows NT™ 4.0 operating system, there exists a monitoring facility called a "system log event viewer" in which a log of events occurring within the platform is recorded into an event log data file which can be inspected by a system administrator using the windows NT operating system software. This facility goes some way to enabling a system administrator to security monitor pre-selected events. The event logging function in the Windows NT™ 4.0 operating system provides system monitoring.

[0008] In terms of overall security of a computer plat-

form, a purely software based system is vulnerable to attack, for example by viruses of which there are thousands of different varieties. Several proprietary virus finding and correcting applications are known, for example the Dr Solomons ™ virus toolkit program The Microsoft Windows NT™ 4.0 software includes a virus guard software, which is preset to look for known viruses. However, virus strains are developing continuously, and the virus guard software will not give reliable protection against newer unknown viruses. New strains of virus are being developed and released into the computing and internet environment on an ongoing basis.

[0009] Further, prior art monitoring systems for computer entities focus on network monitoring functions, where an administrator uses network management software to monitor performance of a plurality of network computers. In these known systems, trust in the system does not reside at the level of individual trust of each hardware unit of each computer platform in a system

Summary of the Invention

[0010] One object of the present invention is to provide a computing entity in which a third party user can have a high degree of confidence that the computing entity has not been corrupted by an external influence, and is operating in a predictable and known manner.

[0011] Another object of the present invention is to simplify a task of judging whether a trustworthiness of a computing entity is sufficient to perform a particular task or set of tasks or type of task.

[0012] In specific implementations of the present invention, a computing entity is capable of residing in a plurality of distinct operating states. Each operating state can be distinguished from other operating states using a set of integrity metrics designed to distinguish between those operating states.

[0013] According to first aspect of the present invention there is provided a computing entity comprising:

a computer platform comprising a plurality of physical and logical resources including a first data processor and a first memory means;

a monitoring component comprising a second data processor and a second memory means;

wherein, said computer platform is capable of operating in a plurality of different states, each said state utilising a corresponding respective set of individual ones of said physical and logical resources;

wherein said monitoring component operates to determine which of said plurality of states said computer platform operates in.

[0014] Preferably a said memory means contains a set of instructions for configuration of said plurality of physical and logical resources of said computer platform into said pre-determined state.

[0015] Preferably exit of said computer platform from said pre-determined state is monitored by said monitoring component.

[0016] A BIOS file may be provided within the monitoring component itself. By providing the BIOS file within the monitoring component, the BIOS file may be inherently trusted.

[0017] In an alternative embodiment, said computer platform may comprise an internal firmware component configured to compute a digest data of a BIOS file data stored in a predetermined memory space occupied by a BIOS file of said computer platform.

[0018] According to second aspect of the present invention there is provided a method of activating a computing entity comprising a computer platform having a first data processing means and a first memory means and a monitoring component having a second data processing means and a second memory means, into an operational state of a plurality of pre-configured operational states into which said computer platform can be activated, said method comprising the steps of:

selecting a state of said plurality of pre-configured operational states into which to activate said computer platform;

activating said computer platform into said selected state according to a set of stored instructions; and

wherein said monitoring component monitors activation into said selected state by recording data describing which of said plurality of pre-configured states said computer platform is activated into.

[0019] Said monitoring component may continue to monitor said selected state after said computer platform has been activated to said selected state

[0020] Said monitoring component may generate a state signal in response to a signal input directly to said monitoring component by a user of said computing entity, said state signal containing data describing which said state said computer platform has entered.

[0021] In one embodiment, said set of stored instructions which allow selection of said state may be stored in a BIOS file resident within said monitoring component. Once selection of a said state has been made, activation of the state may be carried out by a set of master boot instructions which are themselves activated by the BIOS

[0022] Preferably the method comprises the step of generating a menu for selection of a said pre-configured state from said plurality of pre-configured states.

[0023] The method may comprise the step of generating a user menu displayed on a user interface for selection of a said pre-configured state from said plurality of pre-configured states, and said step of generating a state signal comprises generating a state signal in response to a user input accepted through said user interface.

[0024] Alternatively, the predetermined state may be automatically selected by a set of instructions stored on a smartcard, which selects a state option generated by said BIOS. The selection of states may be made automatically via a set of selection instructions to instruct said BIOS to select a state from said set of state options generated by said BIOS.

[0025] Said step of monitoring a said state may comprise:

immediately before activating said computer platform, creating by means of a firmware component a digest data of a first pre-allocated memory space occupied by a BIOS file of said computer platform;

writing said digest data to a second pre-allocated memory space to which only said firmware component has write access; and

said monitoring component reading said digest data from said second pre-allocated memory space.

[0026] Said step of monitoring a said state into which said computer platform is activated may comprise:

executing a firmware component to compute a digest data of a BIOS file of said computer platform;

writing said digest data to a predetermined location in said second memory means of said monitoring component.

[0027] Said step of activating said computer platform into said selected state may comprise:

at a memory location of said first memory means, said location occupied by a BIOS file of said computer platform, storing an address of said monitoring component which transfers control of said first processor to said monitoring component;

storing in said monitoring component a set of native instructions which are accessible immediately after reset of said first processor, wherein said native instructions instruct said first processor to calculate a digest of said BIOS file and store said digest data in said second memory means of said monitoring component; and

said monitoring component passing control of said activation process to said BIOS file, once said digest data is stored in said second memory means.

[0028] Said step of monitoring said state into which said computer platform is activated may comprise:

after said step of activating said computer platform into said selected state, monitoring a plurality of logical and physical components to obtain a first set of metric data signals from those components, said metric data signals describing a status and condition of said components;

comparing said first set of metric data signals determined from said plurality of physical and logical components of said computer platform, with a set of pre-recorded metric data stored in a memory area reserved for access only by said monitoring component; and

comparing said first set of metric data signals obtained directly from said plurality of physical and logical components with said set of pre-stored metric data signals stored in said reserved memory area.

[0029] According to a third aspect of the present invention there is provided a method of operating a computing entity comprising a computer platform having a first data processing means and a first memory means, and a monitoring component having a second data processing means and a second memory means, such that said computer platform enters one of a plurality of possible pre-determined operating states said method comprising the steps of:

in response to an input from a user interface, generating a state signal, said state signal describing a selected state into which said computer platform is to be activated into;

activating said computer platform into a pre-determined state, in which a known set of physical and logical resources are available for use in said state and known processes can operate in said state;

from said pre-determined state, entering a configuration menu for reconfiguration of said monitoring component; and

modifying a configuration of said monitoring component by entering data via a user interface in accordance with an instruction set comprising said configuration menu.

[0030] Said step of entering said monitoring component configuration menu may comprise:

entering a confirmation key signal directly into said monitoring component, said confirmation key signal generated in response to a physical activation of a confirmation key.

[0031] Said step of entering said monitoring component configuration menu may comprise entering a password to said trusted component via a user interface.

[0032] According to a fourth aspect of the present invention there is provided a method of operation of a

computing entity comprising a monitoring component having a first data processing means and a first memory means, and a computer platform having a second data processing means and a second memory means, said method comprising the steps of:

entering a first state of said computer entity, wherein in said first state are available a plurality of pre-selected physical and logical resources;

commencing a user session in said first state, in which said user session a plurality of data inputs are received by said computer platform, said second data processing means performing data processing on said received data;

reconfiguring said plurality of physical and logical resources according to instructions received in said session;

generating a session data describing a configuration of said physical and logical resources;

generating a plurality of user data resulting from processes operating within said session;

storing said user data;

storing session data;

exiting said session; and

exiting said computer platform from said state

[0033] Said method may further comprise the step of reconfiguring said monitoring component during said user session in said first state. Thus, the monitoring component may be reconfigured from a trusted state of the computer platform.

Brief Description of the Drawings

[0034] For a better understanding of the invention and to show how the same may be carried into effect, there will now be described by way of example only, specific embodiments, methods and processes according to the present invention with reference to the accompanying drawings in which:

Fig. 1 illustrates schematically a computer entity according to first specific embodiment of the present invention;

Fig. 2 illustrates schematically connectivity of selected components of the computer entity of Fig. 1;

Fig. 3 illustrates schematically a hardware architecture of components of the computer entity of Fig. 1;

Fig. 4 illustrates schematically an architecture of a trusted component comprising the computer entity of Fig. 1;

Fig. 5 illustrates schematically a logical architecture of the computer entity, divided into a monitored user space resident on a computer platform and a trusted space resident on the trusted component;

Fig. 6 illustrates schematically a set of physical and logical resources comprising the computer entity, wherein different combinations of usage and accessibility to the individual physical and logical resources corresponds with operation in different states of the computing entity;

Fig. 7 illustrates schematically an example of a state diagram illustrating a set of states into which the computing entity can be placed, and processes for entry and exit from those states;

Fig. 8 illustrates schematically a use model followed by a user of the computing entity for entry and exit from individual states of the computing entity;

Fig. 9 illustrates schematically steps of a process for entry into a trusted state;

Fig. 10 illustrates schematically a first mode of operation of the computing entity in a trusted state, in which a first session is carried out by a user;

Fig. 11 illustrates schematically a second session carried out in a trusted state, wherein the second session is carried out after closure of the first session; and

Fig. 12 illustrates schematically a second mode of operation of the computer entity in which reconfiguration of a trusted component may be made by a user.

Detailed Description of the Best Mode for Carrying Out the Invention

[0035] There will now be described by way of example the best mode contemplated by the inventors for carrying out the invention. In the following description numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent however, to one skilled in the art, that the present invention may be practiced without limitation to these specific details. In other instances, well known methods and structures have not been described in detail so as not to unnecessarily obscure the present invention.

[0036] Specific embodiments of the present invention comprise a computer platform having a processing

means and a memory means, and which is physically associated with a component, known herein after as a "trusted component" which monitors operation of the computer platform by collecting metrics data from the computer platform, and which is capable of verifying to third party computer entities interacting with the computer platform to the correct functioning of the computer platform.

[0037] Two computing entities each provisioned with such a trusted component, may interact with each other with a high degree of 'trust'. That is to say, where the first and second computing entities interact with each other the security of the interaction is enhanced compared to the case where no trusted component is present, because:

- A user of a computing entity has higher confidence in the integrity and security of his/her own computer entity and in the integrity and security of the computer entity belonging to the other computing entity.
- Each entity is confident that the other entity is in fact the entity which it purports to be.
- Where one or both of the entities represent a party to a transaction, e.g. a data transfer transaction, because of the in-built trusted component, third party entities interacting with the entity have a high degree of confidence that the entity does in fact represent such a party.
- The trusted component increases the inherent security of the entity itself, through verification and monitoring processes implemented by the trusted component.
- The computer entity is more likely to behave in the way it is expected to behave.

[0038] In this specification, the term "trusted" when used in relation to a physical or logical component, is used to mean a physical or logical component which always behaves in an expected manner. The behavior of that component is predictable and known. Trusted components have a high degree of resistance to unauthorized modification.

[0039] In this specification, the term "computer platform" is used to refer to at least one data processor and at least one data storage means, usually but not essentially with associated communications facilities e.g. a plurality of drivers, associated applications and data files, and which may be capable of interacting with external entities e.g. a user or another computer entity, for example by means of connection to the Internet, connection to an external network, or by having an input port capable of receiving data stored on a data storage medium, e.g. a CD ROM, floppy disk, ribbon tape or the like. The term "computer platform" encompasses the main data processing and storage facility of a computer entity.

[0040] Referring to Fig. 1 herein, there is illustrated schematically one example of a computer entity according to a specific implementation of the present invention. Referring to Fig. 2 of the accompanying drawings, there is illustrated schematically physical connectivity of some of the components of the trusted computer entity of Fig. 1. Referring to Fig. 3 herein, there is illustrated schematically an architecture of the trusted computer entity of Figs. 1 and 2, showing physical connectivity of components of the entity.

[0041] In general, in the best mode described herein, a trusted computer entity comprises a computer platform consisting of a first data processor, and a first memory means, together with a trusted component which verifies the integrity and correct functioning of the computing platform. The trusted component comprises a second data processor and a second memory means, which are physically and logically distinct from the first data processor and first memory means.

[0042] In the example shown in Figs. 1 to 3 herein, the trusted computer entity is shown in the form of a personal computer suitable for domestic use or business use. However, it will be understood by those skilled in the art that that this is just one specific embodiment of the invention, and other embodiments of the invention may take the form of a palmtop computer, a laptop computer, a server-type computer, a mobile phone-type computer, or the like and the invention is limited only by the scope of the claims herein. In the best mode example described herein, the computer entity comprises a display monitor 100; a keyboard data entry means 101; a casing 102 comprising a motherboard on which is mounted a data processor; one or more data storage means e.g. hard disk drives; a dynamic random access memory; various input and output ports (not illustrated in Fig. 1); a smart card reader 103 for accepting a user's smart card; a confirmation key 104, which a user can activate when confirming a transaction via the trusted computer entity; and a pointing device, e.g. a mouse or trackball device 105; and a trusted component.

[0043] Referring to Fig. 2 herein, there are illustrated some of the components comprising the trusted computer entity, including keyboard 101, which incorporates confirmation key 104 and smart card reader 103; a main motherboard 200 on which is mounted first data processor 201 and trusted component 202, an example of a hard disc drive 203, and monitor 100. Additional components of the trusted computer entity, include an internal frame to the casing 102, housing one or more local area network (LAN) ports, one or more modem ports, one or more power supplies, cooling fans and the like (not shown in Fig. 2).

[0044] In the best mode herein, as illustrated in Fig. 3 herein, main motherboard 200 is manufactured comprising a first data processor 201, and preferably a permanently fixed trusted component 202, a local memory

device 300 to the first data processor, the local memory device being a fast access memory area, e.g. a random access memory; a BIOS memory area 301; smart card interface 305; a plurality of control lines 302; a plurality of address lines 303; a confirmation key interface 306; and a data bus 304 connecting the processor 201, trusted component 202, memory area 300, a BIOS memory component 301 and smart card interface 305. A hardware random number generator RNG 309 is also able to communicate with the processor 201 using the bus 304.

[0045] External to the motherboard and connected thereto by data bus 304 are provided the one or more hard disk drive memory devices 203, keyboard data entry device 101, pointing device 105, e.g. a mouse, trackball device or the like, monitor device 100; smart card reader device 103 for accepting a smart card device as described previously; the disk drive(s), keyboard, monitor, and pointing device being able to communicate with processor 201 via said data bus 304; and one or more peripheral devices 307, 308, for example a modem, printer scanner or other known peripheral device.

[0046] To provide enhanced security confirmation key switch 104 is hard wired directly to confirmation key interface 306 on motherboard 200, which provides a direct signal input to trusted component 202 when confirmation key 104 is activated by a user such that a user activating the confirmation key sends a signal directly to the trusted component, by-passing the first data processor and first memory means of the computer platform.

[0047] In one embodiment the confirmation key may comprise a simple switch. Confirmation key 104, and confirmation key driver 306 provide a protected communication path (PCP) between a user and the trusted component, which cannot be interfered with by processor 201, which by-passes data bus 304 and which is physically and logically unconnected to memory area 300 or hard disk drive memory device(s) 203.

[0048] Trusted component 202 is positioned logically and physically between monitor 100 and processor 201 of the computing platform, so that the trusted component 202 has direct control over the views displayed on monitor 100 which cannot be interfered with by processor 201.

[0049] The trusted component lends its identity and trusted processes to the computer platform and the trusted component has those properties by virtue of its tamper-resistance, resistance to forgery, and resistance to counterfeiting. Only selected entities with appropriate authentication mechanisms are able to influence the processes running inside the trusted component. Neither a user of the trusted computer entity, nor anyone or any entity connected via a network to the computer entity may access or interfere with the processes running inside the trusted component. The trusted component has the property of being "inviolate".

[0050] Smart card reader 103 is wired directly to smart card interface 305 on the motherboard and does not connect directly to data bus 304. Alternatively, smart card reader 103 may be connected directly to data bus 304. On each individual smart card may be stored a corresponding respective image data which is different for each smart card. For user interactions with the trusted component, e.g. for a dialogue box monitor display generated by the trusted component, the trusted component takes the image data from the user's smart card, and uses this as a background to the dialogue box displayed on the monitor 100. Thus, the user has confidence that the dialogue box displayed on the monitor 100 is generated by the trusted component. The image data is preferably easily recognizable by a human being in a manner such that any forgeries would be immediately apparent visually to a user. For example, the image data may comprise a photograph of a user. The image data on the smart card may be unique to a person using the smart card.

[0051] Referring to Fig. 4 herein, there is illustrated schematically an internal architecture of trusted component 202. The trusted component comprises a processor 400, a volatile memory area 401; a non-volatile memory area 402; a memory area storing native code 403; and a memory area storing one or a plurality of cryptographic functions, 404, the non-volatile memory 402, native code memory 403 and cryptographic memory 404 collectively comprising the second memory means herein before referred to.

[0052] Trusted component 202 comprises a physically and logically independent computing entity from the computer platform. In the best mode herein, the trusted component shares a motherboard with the computer platform so that the trusted component is physically linked to the computer platform. In the best mode, the trusted component is physically distinct from the computer platform, that is to say it does not exist solely as a sub-functionality of the data processor and memory means comprising the computer platform, but exists separately as a separate physical data processor 400 and separate physical memory area 401, 402, 403, 404. By providing a physically present trusted component separate from a main processor of the computer entity, the trusted component becomes harder to mimic or forge through software introduced onto the computer platform. Another benefit which arises from the trusted component being physical, separate from the main processor of the platform, and tamper resistant is that the trusted component cannot be physically subverted by a local user, and cannot be logically subverted by either a local user or a remote entity. Programs within the trusted component are pre-loaded at manufacture of the trusted component in a secure environment. The programs cannot be changed by users, but may be configured by users, if the programs are written to permit such configuration. The physicality of the trusted component, and the fact that the trusted component is not configurable by the user enables the user to have confidence in the inherent integrity of the trusted component, and

therefore a high degree of "trust" in the operation and presence of the trusted component on the computer platform.

[0053] Referring to Fig. 5 herein, there is illustrated schematically a logical architecture of the computer entity 500. The logical architecture has a same basic division between the computer platform, and the trusted component, as is present with the physical architecture described in Figs. 1 to 3 herein. That is to say, the trusted component is logically distinct from the computer platform to which it is physically related. The computer entity comprises a user space 501 being a logical space which is physically resident on the computer platform (the first processor and first data storage means) and a trusted component space 502 being a logical space which is physically resident on the trusted component 202. In the user space 501 are one or a plurality of drivers 503, one or a plurality of applications programs 504, a file storage area 505; smart card reader 103; smart card interface 305; and a software agent 506 which operates to perform operations in the user space and report back to trusted component 202. The trusted component space is a logical area based upon and physically resident in the trusted component, supported by the second data processor and second memory area of the trusted component. Confirmation key device 104 inputs directly to the trusted component space 502, and monitor 100 receives images directly from the trusted component space 502. External to the computer entity are external communications networks e.g. the Internet 507, and various local area networks, wide area networks 508 which are connected to the user space via the drivers 503 which may include one or more modem ports. External user smart card 509 inputs into smart card reader 103 in the user space.

[0054] In the trusted component space, are resident the trusted component itself, displays generated by the trusted component on monitor 100; and confirmation key 104, inputting a confirmation signal via confirmation key interface 306.

[0055] In the best mode for carrying out the invention, the computing entity has a plurality of modes of operation, referred to herein as operating states. Different ones of the plurality of operating states allow the computing entity to perform different sets of tasks and functionality. In some of the individual states, complex operations can be carried out with a large number of degrees of freedom, and complexity. In other operating states, there are more restrictions on the behavior of the computing entity.

[0056] The level of 'trust' which can be placed on the computing entity when operating in each of the plurality of different states is related to:

- The number of different operations which can be carried out in a particular state
- The complexity of operations which can be carried out in a particular state.
- A number of other states into which the computing entity can move from the particular state, without re-booting the computing entity
- A number of different states from which the particular state can be arrived at, without re-booting the computing entity.
- The connectivity of the computing entity when in the particular state, that is to say, how many other computing entities or devices the entity is connectable to, e.g. over the internet, a wide area network, or a local area network.
- Restrictions on input of data from an external source, e.g. another computing entity, a floppy disk, a CD ROM, a modem, a LAN port, or the like.
- Restrictions on output of data from the particular state to other computing entities, e.g. whether data can be saved to a CD writer, floppy disc drive, or exported through an interface to a further computer entity over the internet, a local area network, or a wide area network.
- An amount of, and a reliability of, internal monitoring processes within the computer entity which occur in the particular state; that is to say, the amount and reliability of a set of metrics applied by the trusted component when in that state.
- A number of checks which need to be made before a user can enter the particular state.
- A difficulty of bypassing one or a plurality of checks which need to be made before a user can enter the particular state.
- A difficulty of overcoming, without bypassing, one or a plurality of checks which are made before a user of the computer entity can enter the computing entity into the particular state

[0057] The trust placed in the computer entity is composed of two separate parts;

- The trust placed in the trusted component itself
- The certainty with which the trusted component can verify operation of the computer entity.

[0058] As described herein, levels or degrees of trust placed in the computer entity are determined as being relative to a level of trust which is placed in the trusted component. Although the amount of trust in a computer entity is related to many factors, a key factor in measuring that trust are the types, extent and regularity of integrity metric checks which the trusted component itself carries out on the computer entity.

[0059] The trusted component is implicitly trusted. The trusted component is embedded as the root of any trust which is placed in the computing platform and the computing platform as a whole cannot be any more trusted than the amount of trust placed in the trusted component.

[0060] By virtue of the trusted component monitoring operations of the computer platform, the trust placed in

the trusted component can be extended to various parts of the computer platform, with the level and extent of trust placed in individual areas of the computer platform, being dependent upon the level and reliability with which the trusted component can monitor that particular area of the computing platform.

[0061] Since the trusted areas of the computing platform are dependent upon the frequency, extent, and thoroughness with which the trusted component applies a set of integrity metric measurements to the computer platform, if the trusted component does not comprehensively measure all measurable aspects of the operation of the computing platform at all times, then the level of trust placed in individual parts of the computer platform will form a subset of the overall trust placed in the trusted component itself. If the computing entity supports only a limited number of integrity metrics, a user of the equipment, including a third party computing entity, is restricted in its ability to reason about the level of trust which can be placed in the computing entity.

[0062] Although various islands of the computer platform are trusted at various levels, depending upon the integrity metrics which are applied by the trusted component for measuring those areas of the computer platform, the level of trust placed in the computer platform as a whole is not as high as that which is inherent in the trusted component. That is to say, whilst the trusted component space 502 is trusted at a highest level, the user space 501 may comprise several regions of various levels of trust. For example, applications programs 504 may be relatively untrusted. Where a user wishes to use the computer entity for an operation which involves a particularly high degree of confidentiality or secrecy, for example working on a new business proposal, setting pay scales for employees or equally sensitive operations, then the human user may become worried about entering such details onto the computer platform because of the risk that the confidentiality or secrecy of the information will become compromised. The confidential information must be stored in the computing entity, and islands of high trust may not extend over the whole computing platform uniformly and with the same degree of trust. For example, it may be easier for an intruder to access particular areas or files on the computing platform compared with other areas or files.

[0063] Additionally, a user may wish to instruct the trusted component to perform certain functions, this poses the problem that all the commands to instruct the trusted component must pass through the computer platform, which is at a lower level of trust than the trusted component itself. Therefore, there is a risk of the commands to the trusted component becoming compromised during their passage and processing through the computer platform.

[0064] According to specific implementations of the present invention, the computer entity may enter a plurality of different states, each state having a corresponding respective level of trust, wherein the individual levels of trust corresponding to different states may be different from each other.

[0065] Referring to Fig. 6, there is illustrated schematically a set of physical and logical resources available to the computing entity. In the general case, the computing entity comprises a plurality of input/output devices 600 for communicating with other computing entities, examples of such devices including a modem, a local area network port, an Ethernet card, a hard disk drive 203, a floppy disk drive, and a smart card reader device 103; a plurality of memory areas 601-603, resident on the hard disk 203, or ram 300; one or a plurality of operating systems 604-606, and one or a plurality of application programs 607-609.

[0066] In this specification, by the term "state" when used in relation to a computing entity, it is meant a mode of operation of the computing entity in which a plurality of functions provided by the computing platform may be carried out. For example in a first state, the computing entity may operate under control of a first operating system, and have access to a first set of application programs, a first set of files, and a first set of communications capabilities, for example modems, disk drives, local area network cards, e.g. Ethernet cards. In a second state, the computing platform may have access to a second operating system, a second set of applications, a second set of data files and a second set of input/output resources. Similarly, for successive third, fourth states up to a total number of states into which the computing entity can be set. There can be overlap between the facilities available between two different states. For example, a first and second state may use a same operating system, whereas a third state may use a different operating system.

[0067] Referring to Fig. 7 herein, there is illustrated schematically a state diagram representing a plurality of states into which the computing entity may be placed. In principle, there is no limit to the number of different states which the computing entity may be placed, but in the example shown in Fig. 7 three such states are shown. In the example of Fig. 7, the computing entity may be placed into a first, trusted state 700, a second state 701 being a general purpose untrusted state and a third state 702 being a general purpose untrusted state. In the general case, the computing entity can reside in a plurality of different states, each having a corresponding respective level of trust.

[0068] Trusted state 700 is distinguished from the second and third states 701, 702 by virtue of the way in which the trusted state can be accessed. In one option, trusted state 700 can only be accessed by reference to the trusted component 202. However, in the preferred best mode implementation entry into the trusted state need not be controlled by the trusted component. To access the trusted state, a user may turn on the computing entity, that is to say turn on the power supply to the computing entity in a turn on process 703. Upon turning on the power supply, the computing entity boots up via the

BIOS file 301 in process 704, from a routine contained in the computer BIOS. The computing entity may enter either the trusted state 700, the second state 701, or the third state 702, depending upon how the BIOS file is configured. In the best mode herein, a user of the computer entity has the option, provided as a menu display option on monitor 100 during boot up of the computer entity, or as a selectable option presented as a screen icon, when in any state, to enter either the trusted state 700, or one of the other states 701, 702 by selection. For example on turn on , the BIOS may be configured to default boot up in to the second state 701. Once in the second state, entry into a different state 700 may require a key input from a user, which may involve entry of a password, or confirmation of the users identity by the user entering their smart card into smart card reader 103.

[0069] Once the computing entity has entered a state other than the trusted state, e.g. the second state 701 or third state 702, then from those states the user may be able to navigate to a different state. For example the user may be able to navigate from the second state 701 to the third state 702 by normal key stroke entry operations on the keyboard, by viewing the monitor and using a pointing device signal input, usually with reference back to the BIOS. This is shown schematically as select new state process 705.

[0070] In order to enter the trusted state 700, the computer entity must be either booted up for the first time after turn on process 704, or re-booted via the BIOS in re-boot process 705. Re-boot process 705 is very similar to boot up process 704 except that it can be entered without having to turn the power of the computing entity off and then on again. To leave the trusted state 700, the computing entity must again refer to the BIOS 704 which involves automatic monitoring by the trusted component 202 in monitor process 706. Similarly, re-booting via the BIOS in process 705 involves automatic monitoring by the trusted component in monitoring process 706.

[0071] To leave the trusted state 700, the trusted state can only be left either by turning the power off in power down process 707, or by re-booting the computing entity in re-boot process 705. Re-booting the BIOS in re-boot process 705 involves automatic monitoring by the trusted component 706. Once the trusted state is left, it is not possible to re-enter the trusted state without either re-booting the computing entity, in re-boot process 705, or booting up the computing entity after a power down in process 704, both of which involve automatic monitoring by the trusted component in monitoring process 706.

[0072] Referring to Fig. 8 herein, there is illustrated schematically a use model followed by a user of the computer entity navigating through one or more states. In step 800, after turning on a power supply to the computing entity, the computer boots up via the BIOS program. The boot process is very similar to re-booting the computer from an existing state. In each case, control of microprocessor 201 is seized by the BIOS component 301. The trusted component 202 measures a set of integrity metric signals from the BIOS 301, to determine a status of the BIOS 301. In step 801, the graphical user interface displays a menu option for entry into a plurality of different states. One of the states displayed on the menu is a trusted state as described herein before. The user manually selects a state in which to enter by using the keyboard or pointing device of the graphical user interface, for example by clicking a pointer icon over a state icon displayed on the graphical user interface. Alternatively, an automatic selection of a state may be made by a smartcard or via a network connection from state selection options generated by the BIOS. After selection of a state, the BIOS loads a program which loads a selected operating system corresponding with the state. A different load program is used for each of the plurality of different possible states. The trusted component measures that program in broadly a similar way to the way in which it measures the BIOS, so that the trusted component can record and determine which state has been loaded. When an external entity requests that the trusted component supplies integrity metrics, the trusted component supplies both the BIOS metrics and the loaded program metrics. In step 802, the computing entity enters the selected state. Once in the selected state, the user has access to a set of physical and logical resources in that state. For example, in a relatively insecure state, the user may have full internet access through a modem device comprising the computing entity, may have full access to one or a plurality of hard disk drives or CD readers/writers, and may have full access to a floppy disk drive, as well as having access to a plurality of pre-loaded commercially available applications programs. On the other hand, if the user selects a trusted state having a relatively high level of trust, in that state the user may have available a single operating system, a limited set of applications, for example a word processor, accounts package, or database, and use of a printer device, but in that state, use of a hard disk drive, a floppy disk drive, or the internet may be restricted. Each selection of a separate state into which the computer may be booted may be pre-configured by configuration of the BIOS component 301. A choice of states is presented by the BIOS to a user. Once a state is selected, the BIOS causes the selected state to load by calling up an operating system loading program to load that state. The states themselves are pre-configured by the loading and the relevent operating system. For entry into trusted states, entry into those states is via operation of the BIOS component 301, and including monitoring by the trusted component in monitoring process 706. In order to enter a trusted state, a user must boot or re-boot the computer platform in step 804. Similarly, to exit from a trusted state, the user must also boot or re-boot the computing entity in step 804. To navigate from a state having a lower trust level, for example the second

state (701), or the third state (702), the user may navigate from that state to another state in step 805, which, in the best mode involves re-booting of the computing entity via the BIOS.

[0073] Referring to Fig. 9 herein there is illustrated schematically process steps carried out by the computing entity for entering a state via boot process 704 or reboot process 705.

[0074] In step 900, the computer enters a boot up routine, either as a result of a power supply to the computing entity being turned on, or as a result of a user inputting a reset instruction signal, for example by clicking a pointer icon over a reset icon displayed on the graphical user interface, giving rise to a reset signal. The reset signal is received by the trusted component, which monitors internal bus 304. The BIOS component 301 initiates a boot-up process of the computer platform in step 901. Trusted component 202 proceeds to make a plurality of integrity checks on the computer platform and in particular checks the BIOS component 301 in order to check the status of the computer platform. Integrity checks are made by reading a digest of the BIOS component. The trusted component 202 acts to monitor the status of the BIOS, and can report to third party entities on the status of the BIOS, thereby enabling third party entities to determine a level of trust which they may allocate to the computing entity.

[0075] There are several ways to implement integrity metric measurement of the BIOS. In each case, the trusted component is able to obtain a digest of a BIOS file very early on in the boot up process of the computer platform. The following are examples

- The BIOS component may be provided as part of the trusted component 202, in which the architecture illustrated in Fig. 3 herein is modified such that BIOS 301 resides within trusted component 202.
- The first processor 201 of the computer platform may execute immediately after reset, an internal firmware component which computes a digest over a preset memory space occupied by a BIOS file. The first processor writes the digest to a preset memory space to which only the firmware component is able to write to that memory space. The first processor reads from the BIOS file in order to boot the computer platform. At any time afterwards, the trusted component reads data from a preset location within the memory space to obtain a BIOS digest data.
- The trusted component may be addressed at a memory location occupied by BIOS 301, so that the trusted component contains a set of first native instructions which are accessed after reset of the first processor 201. These instructions cause the first processor 201 of the computer platform to calculate a digest of the BIOS, and store it in the trusted component. The trusted component then passes control to the BIOS 301 once the digest of the BIOS is stored in the trusted component.
- The trusted component may monitor a memory control line and a reset line and verify that the BIOS component 301 is the first memory location accessed after the computer platform resets. At some stage in the boot process, the BIOS passes control to the trusted component and the trusted component causes the first processor of the computer platform to compute a digest of the BIOS and return the digest to the trusted component. The process of computing the digest and writing the result to the trusted component must be atomic. This action may be started by the trusted component, causing the computer platform's processor to read a set of native instructions from the trusted component which causes the processor to compute a digest over a memory space occupied by the BIOS, and to write the digest data to the memory space occupied by the trusted component. Alternatively, this action could be started by the trusted component causing the first processor of a platform to execute an instruction, where the processor computes a digest over a preset memory space occupied by the BIOS and writes the digest to a preset memory space occupied by the trusted component.
- A loading program for loading a selected operating system is itself loaded by the BIOS program. Integrity metrics of the operating system loading program are also measured by computing a digest of the loading program.

[0076] In one embodiment, trusted component 202 may interrogate individual components of the computer platform, in particular hard disk drive 203, microprocessor 201, and RAM 301, to obtain data signals directly from those individual components which describe the status and condition of those components. Trusted component 202 may compare the metric signals received from the plurality of components of the computer entity with the pre-recorded metric data stored in a memory area reserved for access by the trusted components. Provided that the signals received from the components of the computer platform coincide with and match those of the metric data stored within the memory, then the trusted component 202 provides an output signal confirming that the computer platform is operating correctly. Third parties, for example, other computing entities communicating with the computing entity may take the output signal as confirmation that the computing entity is operating correctly, that is to say is trusted.

[0077] In step 903 BIOS generates a menu display on monitor 100 offering a user a choice of state options, including a trusted state 700. The user enters details of which state is to be entered by making key entry to the graphical user interface or data entry using a pointing device, e.g. mouse 105. The BIOS receives key inputs from a user which instruct a state in to which to boot in step 904. The trusted component may also require a

separate input from confirmation key 104 requiring physical activation by a human user, which bypasses internal bus 304 of the computer entity and accesses trusted component 202 directly, in addition to the user key inputs selecting the state. Once the BIOS 301 has received the necessary key inputs instructing which state is required, the processing of the set of configuration instructions stored in BIOS 301 occurs by microprocessor 201, and instructs which one of a set of state options stored in the BIOS file, the computer platform will configure itself into. Each of a plurality of state selections into which the computer platform may boot may be stored as separate boot options within BIOS 301, with selection of the boot option being controlled in response to keystroke inputs or other graphical user inputs made by a user of the computing entity. Once the correct routine of BIOS file 301 is selected by the user, then in step 906, the BIOS file then releases control to an operating system load program stored in a memory area of the computer platform, which activates boot up of the computer platform into an operating system of the selected state. The operating system load program contains a plurality of start up routines for initiating a state, which include routines for starting up a particular operating system corresponding to a selected state. The operating load program boots up the computer platform into the selected state. The operating system measures the metrics of the load program which is used to install the operating system, in step 907. Once in the selected state, trusted component 202 continues, in step 908, to perform on an ongoing continuous basis further integrity check measurements to monitor the selected state continuously, looking for discrepancies, faults, and variations from the normal expected operation of the computer platform within that state. Such integrity measurements are made by trusted component 202 sending out interrogation signals to individual components of the computer platform, and receiving response signals from the individual components of the computer platform, which response signals the trusted component may compare with a predetermined preloaded set of expected response signals corresponding to those particular states which are stored within the memory of the trusted component, or the trusted component 202 compares the integrity metrics measured from the computer platform in the selected state with the set of integrity metrics initially measured as soon as the computer platform enters the selected state, so that on an ongoing basis any changes to the integrity metrics from those initially recorded can be detected.

[0078] During the boot up procedure, although the trusted component monitors the boot up process carried out by the BIOS component, it does not necessarily control the boot up process. The trusted component acquires a value of the digest of the BIOS component 301 at an early stage in the boot up procedure. In some of the alternative embodiments, this may involve the trusted component seizing control of the computer platform before boot up by the BIOS component commences. However, in alternative variations of the best mode implementation described herein, it is not necessary for the trusted component to obtain control of the boot up process, but the trusted component does monitor a computer platform, and in particular the BIOS component 301. By monitoring the computer platform, the trusted component stores data which describes which BIOS options have been used to boot up the computer, and which operating system has been selected. The trusted component also monitors the loading program used to install the operating system.

[0079] There will now be described an example of operation of a computer entity within a trusted state in a first specific mode of operation according to the present invention.

[0080] Referring to Figs. 10 and 11 herein, there is illustrated schematically usage of the computing entity in a trusted state, extending over a plurality of user sessions, for example usage of the computing entity over two successive days, whilst turning off or re-booting the computing entity between sessions.

[0081] Referring to Fig. 10 herein, a user boots up the computing entity into a trusted state 700 as herein before described in a first boot process 1000. In the trusted state, the user commences a first session 1001 of usage of the computing entity. Within the session, because the computer platform is booted into the trusted state, a predetermined set of logical and physical resources are available to the user within that trusted state. Typically, this would include access to an operating system and a predetermined selection of applications. The level of trust which applies to the trusted state varies depending upon the number, complexity and reliability of the physical and logical resources available to the user within the trusted state. For example, where the trusted state is configured to use a well-known reliable operating system, for example UNIX, and a reliable word processing package with minimal access to peripheral devices of the computer platform being permitted in the trusted state, for example no access to modems, and access to output data restricted to a single writer drive, e.g. a CD writer, then this may have a relatively high degree of trust. In another trusted state, where more facilities are available, the trust level would be different to that in a trusted state in which more limited access to physical or logical resources. However, each trusted state is characterized in that the access to facilities is predetermined and known and can be verified by trusted component 202. During the first session 1001, a user may call up an application 1002 available in the trusted state, and may enter user data 1003, for example via a keyboard device. The user data 1003 is processed according to the application 1002 in processing operation 1004, resulting in processed output user data 1005. During the course of the session, by virtue of using the computer platform, operating system and applications, the user may have reconfigured the applications and/or operat-

ing system for a specific usage within the session. For example, in a word processor application, documents may have been formatted with certain line spacing, font styles etc. To avoid these settings being lost on leaving the trusted state, such settings comprising session data 1006 may be stored during the session. Similarly, to avoid the effort made by the user during the session being lost, the output user data may be stored during the session. However, the user session 1001 only exists in the trusted state as long as the trusted state exists. Therefore, to avoid loss of settings and data from the first session 1001 in the trusted state 700, the output user data and session data must be stored as stored output user data 1007 and stored session data 1008 respectively before the trusted state can be exited. The stored output user data 1007 and stored session data 1008 may be saved to a device available in the trusted state, for example hard disk drive 203 or a CD reader/writer peripheral for use in a further successive session, or be encrypted and signed and then saved at a remote location, accessed over a network. Preferably, signing of user data and session data is done by the trusted component and/or the user's smartcard. Exit from the trusted states involves closing the first user session 1001, and rebooting the computing entity via re-boot process 705, or powering down the computing entity via power down process 707. In the first user session in the trusted state, processing of user input data occurs, and the output of the process is the output processed data. The output processed data is stored after processing of the data has terminated, and before the session is ended, and before the trusted state is exited.

[0082] Referring to Fig. 11 herein, there is illustrated schematically operation of the computing entity on a second day, in a second session in the same trusted state 700. Between the first and second sessions the trusted state 700 disappears completely, since the computing entity leaves the trusted state 700. On leaving the trusted state 700, apart from the stored output user data and stored session data, the computer platform saves no information concerning the trusted state other than that which is pre-programmed into the BIOS 301 and the loading programs and the trusted component 202. Therefore, for all practical purposes, on power down or re-boot, the trusted state 700 ceases to exist. However, the ability to re-enter the trusted state 700 through a new operation of the boot process or re-boot process remains within the capabilities or the computing entity. The trusted state is entered via a second boot process 1100 as herein before described. Once the trusted state is entered, a second session 1101 commences. Within the second session 1101 the operating system, applications and facilities available from the computer platform are selected from the same set of such physical and logical resources as where available previously for the first session. However, usage of those facilities within the second session may vary according to a user's keystroke instructions. Second session 1101 may effectively comprise a continuation of first session 1001. The user may call up the same application 1002 as previously and may effectively continue the work carried out during the first session in the second session 1101. However, because exiting the trusted state involves the computer platform in complete amnesia of all events which occurred during that trusted state, after the state has been left, if the trusted state is reactivated and the new session is commenced, the application 1002 has no memory of its previous configuration. Therefore, stored output session data 1008 produced at the end of the first session 1001 must be input into the second session 1101 in order to reconfigure the application, to save for example the settings of line spacing and format, and the output user data 1005 stored as stored output user data 1007 must be re-input into the second session 1101 for further work to continue on that data. The stored session data 1008 and user data 1007 may be retrieved from a storage medium, decrypted and authenticated and then loaded into the trusted state, to configure the second session as a continuation of the first session. Preferably, integrity measurement checks are performed by the trusted component on the user data and session data imported from the smartcard or storage medium, before that data is loaded. During the second session 1101, further user data 1102 is input by the user, and the further data is processed together with the stored first output data 1007 according to the application 1002 configured according to the first stored output session data 1008 in process 1103. Processing of the data 1103 during the second session 1101 results in a new output user data 1104. If the application or operating system used in the second session has changed in configuration during the second session, this results in a new session data 1105. As with the first session, in order to close the session without losing the settings of the application program, and operating system, and without losing the benefit of the work carried out during the second session, both the new session data 1105 and the new output user data 1104 need to be stored. These data are stored respectively as a stored new output user data 1106 and a stored new session data 1107.

[0083] At the end of the second session, the session is closed after having saved the work produced in the second session, and the trusted state is exited via a power down process or re-boot process 705, 707. All memory of the trusted state and second session other than that stored as the session data 1107 and stored output user data 1106 is lost from the computer platform.

[0084] It will be appreciated that the above example is a specific example of using a computer in successive first and second sessions on different days. In between use of those sessions, the computing entity may be used in a plurality of different states, for different purposes and different operations, with varying degrees of trust. In operating states which have a lower level of trust, for example the second and third states (being 'untrusted' states) the computer entity will not lose memory of this

data configuration between transitions from state to state. According to the above method of operation, the trusted state 700 may be activated any number of times, and any number of sessions carried out. However, once the trusted state is exited, the trusted state has no memory of previous sessions. Any configuration of the trusted state must be by new input of data 1003, 1102, or by input of previously stored session data or user data 1007, 1008, 1106, 1107.

[0085] In the above described specific implementations, specific methods, specific embodiments and modes of operation according to the present invention, a trusted state comprises a computer platform running a set of processes all of which are in a known state. Processes may be continuously monitored throughout a session operating in the trusted state, by a trusted component 202.

[0086] Referring to Fig. 12 herein, there is illustrated schematically a second mode of operation of a trusted state, in which the trusted component itself 202 can be reconfigured by a user. In the second mode of operation, the trusted component stores a predetermined set of data describing metrics which apply when the computer platform is in the trusted state in which the component itself can be reconfigured. A trusted state 1200 is entered as described previously herein through boot process 704 or re-boot process 705. In the trusted state, a user enters a command to call up a trusted component configuration menu in step 1201. The trusted component configuration menu comprises a set of instructions stored in memory and which is only accessible via a trusted state. In order to make changes to the menu, various levels or security may be applied. For example, a user may be required to enter a secure password, for example a password comprising numbers and letters or other characters in step 1202. The trusted component monitors the trusted state from which the trusted component can be reconfigured by comparing measured integrity metrics from the computer platform whilst in the trusted state, with the set of pre-stored integrity metrics which the trusted component stores in its own memory area. The trusted component will not allow a user to reconfigure the trusted component 202 unless the integrity metrics measured by the trusted component when the computer platform is in the trusted state from which the trusted component can be reconfigured match the pre-stored values in the trusted component's own memory, thereby verifying that the computer platform is operating correctly in the trusted state. The trusted component denies a user reconfiguration of the trusted component if the trusted component detects that the measured integrity metrics of the computer platform do not match those predetermined values which are stored in the trusted component's own internal memory, and are those of the trusted state from which the trusted component can be re-configured.

[0087] Additionally, or optionally, the user may be required to insert a smart card into smart card reader 103 in step 1203, following which the trusted component verifies the identity of the user by reading data from the smart card via smart card interface 305. Additionally, the user may be required to input physical confirmation of his or her presence by activation of confirmation key 104 providing direct input into trusted component 202 as described with reference to Fig. 3 herein in step 1204. Data describing the trusted state, for example, which operating system to use, and which applications to use, may be stored on the smart card and used to boot up the computer platform into the trusted state.

[0088] Once the security checks including the password, verification by smart card and/or activation of the confirmation key are accepted by the trusted component, the file configuration menu is displayed on the graphical user interface under control of trusted component 202 in step 1205. Reconfiguration of the trusted component can be made using the menu in step 1206 by the user. Depending upon the level of security applied, which is an implementation specific detail of the trusted component configuration menu, the user may need to enter further passwords and make further confirmation key activations when entering data into the menu itself. In step 1207, the user exits the trusted component reconfiguration menu having reconfigured the trusted component.

[0089] In the trusted component configuration menu, a user may reconfigure operation of the trusted component. For example, a user may change the integrity metrics used to monitor the computer platform.

[0090] By storing predetermined digest data corresponding to a plurality of integrity metrics present in a state inside the trusted component's own memory, this may provide the trusted component with data which it may compare with a digest data of a state into which the computer platform is booted, for the trusted component to check that the computer platform has not been booted into an unauthorized state.

[0091] The trusted component primarily monitors boot up of the computer platform. The trusted component does not necessarily take control of the computer platform if the computer platform boots into an unauthorized state, although optionally, software may be provided within the trusted component which enables the trusted component to take control of the computer platform if the computer platform boots into an unauthorized, or an unrecognized state.

[0092] When in the trusted state, a user may load in new applications to use in that trusted state, provided the user can authenticate those applications for use in the trusted state. This may involve a user entering a signature data of the required application to the trusted component, to allow the trusted component to verify the application by means of its signature when loading the application into the trusted state. The trusted component checks that the signature of the application is the same as the signature which the user has loaded into the trusted component before actually loading the appli-

cation. At the end of a session, the application is lost from the platform altogether. The session in the trusted state exists only in temporary memory, for example random access memory, which is reset when the trusted state is exited.

[0093] In the above described implementations, a version of a computer entity in which a trusted component resides within a video path to a visual display unit have been described. However, the invention is not dependent upon a trusted component being present in a video path to a visual display unit. It will be understood by persons skilled in the art that the above best mode implementations are exemplary of a large class of implementations which can exist according to the invention.

[0094] In the above described best mode embodiment, methods of operation have been described wherein a user is presented with a set of options for selecting a state from a plurality of states, and a user input is required in order to enter a particular desired state. For example a user input may be required to specify a particular type of operating system which is required to be used, corresponding to a state of the computer platform. In a further mode of operation of the specific embodiment, data for selecting a predetermined operating state of the computer platform may be stored on a smart card, which is transportable from computer platform to computer platform, and which can be used to boot up a computer platform into a predetermined required state. The smartcard responds to a set of state selection options presented by a BIOS, and selects one of a plurality of offered choices of state. The BIOS contains the state selections available, and a set of loading programs actually install the various operating systems which provide the states. In this mode of operation, rather than data describing a predetermined state being stored within the first memory area of the trusted component, and the BIOS system obtaining that data from the trusted component in order to boot the computer platform up into a required predetermined state, the information can be accessed from a smart card entered into the smart card reader.

[0095] Using such a smart card pre-configured with data for selecting one or a plurality of predetermined states, a user carrying the smart card may activate any such computing entity having a trusted component and computer platform as described herein into a predetermined state as specified by the user, with a knowledge that the computing entity will retain no record of the state after a user session has taken place. Similarly as described with reference to Figs. 10 and 11 herein, any output user data or configuration data produced during a session may be verified by the smart card, which can be taken away by a user and used to boot up a further different computing entity into the same state, and continue a session on a different computing entity, verifying any information on user data or session data which is to be retrieved, without either computing entity retaining a permanent record of the predetermined state, and without either computing entity retaining any of the processed user data or session configuration data of the predetermined state.

Claims

1. A computing entity comprising:

   a computer platform comprising a plurality of physical and logical resources including a first data processor and a first memory means;

   a monitoring component comprising a second data processor and a second memory means;

   wherein, said computer platform is capable of operating in a plurality of different states, each said state utilising a corresponding respective set of individual ones of said physical and logical resources;
   wherein said monitoring component operates to determine which of said plurality of states said computer platform operates in.

2. The computing entity as claimed in claim 1, wherein a said memory means contains a set of instructions for configuration of said plurality of physical and logical resources of said computer platform into said pre-determined state.

3. The computing entity as claimed in claim 1, in which exit of said computer platform from said pre-determined state is monitored by said monitoring component.

4. The computing entity as claimed in claim 1, wherein said monitoring component includes a BIOS file.

5. The computing entity as claimed in claim 1, wherein said computer platform comprises an internal firmware component configured to compute a digest data of a BIOS file data stored in a predetermined memory space occupied by a BIOS file of said computer platform.

6. A method of activating a computing entity comprising a computer platform having a first data processing means and a first memory means and a monitoring component having a second data processing means and a second memory means, into an operational state of a plurality of pre-configured operational states into which said computer platform can be activated, said method comprising the steps of:

   selecting a state of said plurality of pre-configured operational states into which to activate said computer platform;

activating said computer platform into said selected state according to a set of stored instructions;

wherein said monitoring component monitors activation into said selected state by recording data describing which of said plurality of pre-configured states said computer platform is activated into.

7. The method as claimed in claim 6, wherein said monitoring component continues to monitor said selected state after said computer platform has been activated into said state.

8. The method as claimed in claim 6, wherein said monitoring component generates a state signal in response to a signal input directly to said monitoring component by a user of said computing entity, said state signal indicating which said state said computer platform has entered.

9. The method as claimed in claim 6, wherein said set of stored instructions are stored in a BIOS file resident within said monitoring component.

10. The method as claimed in claim 6, comprising the step of generating a menu for selection of a said pre-configured state from said plurality of pre-configured states.

11. The method as claimed in claim 6, comprising the step of generating a user menu displayed on a user interface for selection of a said pre-configured state from said plurality of pre-configured states, and said step of generating a state signal comprises generating a state signal in response to a user input accepted through said user interface.

12. The method as claimed in claim 7, in which said step of selecting a state of said plurality of pre-configured operational states comprises receiving a selection signal from a smartcard device, said selection signal instructing a BIOS of said computer platform to activate the said computer platform into a said selected state.

13. The method as claimed in claim 6, wherein said step of selecting a state of said plurality of pre-configured operational states comprises receiving a selection message from a network connection, said selection message instructing a BIOS file of said computer platform to activate said computer platform into a said selected state.

14. The method as claimed in claim 6, wherein said step of monitoring a said state comprises:

immediately before activating said computer platform, creating by means of a firmware component a digest data of a first pre-allocated memory space occupied by a BIOS file of said computer platform;

writing said digest data to a second pre-allocated memory space to which only said firmware component has write access; and

said monitoring component reading said digest data from said second pre-allocated memory space.

15. The method as claimed in claim 6, wherein said step of monitoring said state into which said computer platform is activated comprises,

executing a firmware component to compute a digest data of a BIOS file of said computer platform,

writing said digest data to a predetermined location in said second memory means of said monitoring component.

16. The method as claimed in claim 6, wherein said step of activating said computer platform into said selected state comprises,

at a memory location of said first memory means, said location occupied by a BIOS file of said computer platform, storing an address of said monitoring component which transfers control of said first processor to said monitoring component;

storing in said monitoring component a set of native instructions which are accessible immediately after reset of said first processor, wherein said native instructions instruct said first processor to calculate a digest of said BIOS file and store said digest data in said second memory means of said monitoring component; and

said monitoring component passing control of said activation process to said BIOS file, once said digest data is stored in said second memory means.

17. The method as claimed in claim 6, wherein said step of monitoring said state into which said computer platform is activated comprises:

after said step of activating said computer platform into said selected state, monitoring a plurality of logical and physical components to obtain a first set of metric data signals from those components, said metric data signals describ-

ing a status and condition of said components;

comparing said first set of metric data signals determined from said plurality of physical and logical components of said computer platform, with a set of pre-recorded metric data stored in a memory area reserved for access only by said monitoring component; and

comparing said first set of metric data signals obtained directly from said plurality of physical and logical components with said set of pre-stored metric data signals stored in said reserved memory area.

18. A method of operating a computing entity comprising a computer platform having a first data processing means and a first memory means, and a monitoring component having a second data processing means and a second memory means, such that said computer platform enters one of a plurality of possible pre-determined operating states said method comprising the steps of:

in response to an input from a user interface generating a said state signal, said state signal describing a selected state into which said computer platform is to be activated into;

activating said computer platform into a pre-determined state, in which a known set of physical and logical resources are available for use in said state and known processes can operate in said state;

from said pre-determined state, entering a configuration menu for reconfiguration of said monitoring component; and

modifying a configuration of said monitoring component by entering data via a user interface in accordance with an instruction set comprising said configuration menu.

19. The method as claimed in claim 18, wherein said step of entering said monitoring component configuration menu comprises:

entering a confirmation key signal directly into said monitoring component, said confirmation key signal generated in response to a physical activation of a confirmation key.

20. The method as claimed in claim 18, wherein said step of entering said monitoring component configuration menu comprises entering a password to said trusted component via a user interface.

21. A method of operation of a computing entity comprising a monitoring component having a first data processing means and a first memory means, and a computer platform having a second data processing means and a second memory means, said method comprising the steps of:

entering a first state of said computer entity, wherein in said first state are available a plurality of pre-selected physical and logical resources;

commencing a user session in said first state, in which said user session a plurality of data inputs are received by said computer platform, said second data processing means performing data processing on said received data;

reconfiguring said plurality of physical and logical resources according to instructions received in said session;

generating a session data describing a configuration of said physical and logical resources;

generating a plurality of user data resulting from processes operating within said session;

storing said user data;

storing session data;

exiting said session; and

exiting said computer platform from said state.

22. The method as claimed in claim 21, further comprising the step of:

reconfiguring said monitoring component during said user session in said first state.

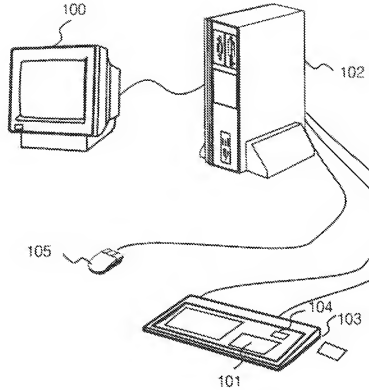


Fig. 1

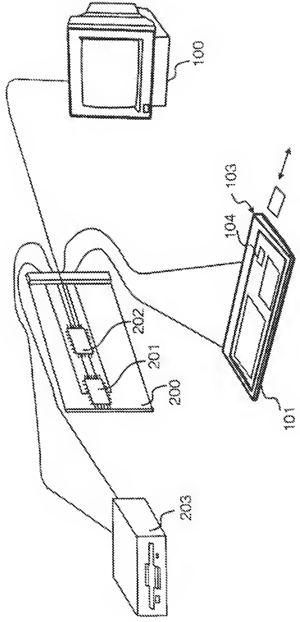
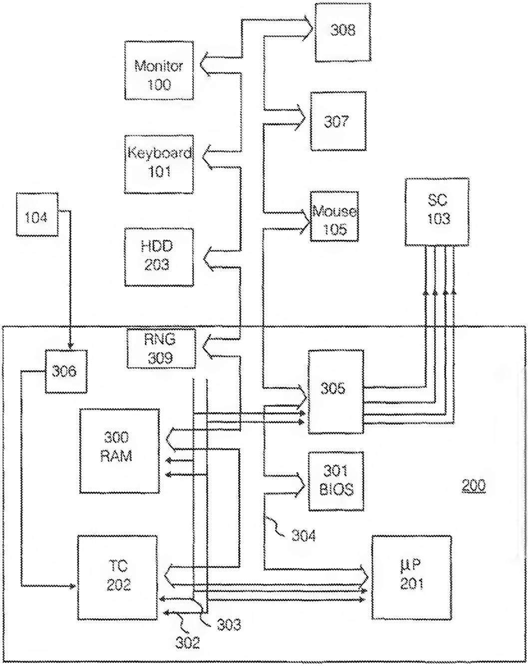


Fig. 2



308
Monitor 100
307
Keyboard 101
104
Mouse 105
SC 103
HDD 203
RNG 309
306
305
300 RAM
301 BIOS
200
304
TC 202
µP 201
303
302

Fig. 3

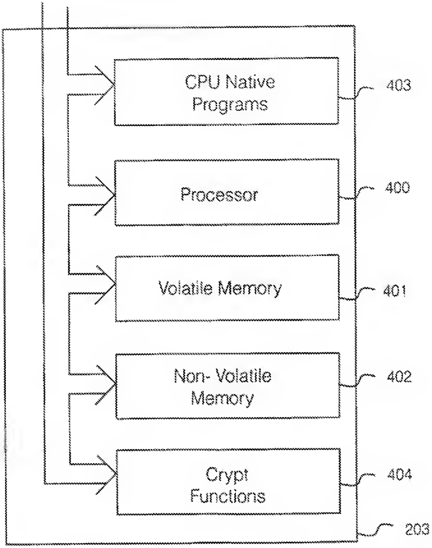

CPU Native Programs
403
Processor
400
Volatile Memory
401
Non- Volatile Memory
402
Crypt Functions
404
203

Fig. 4

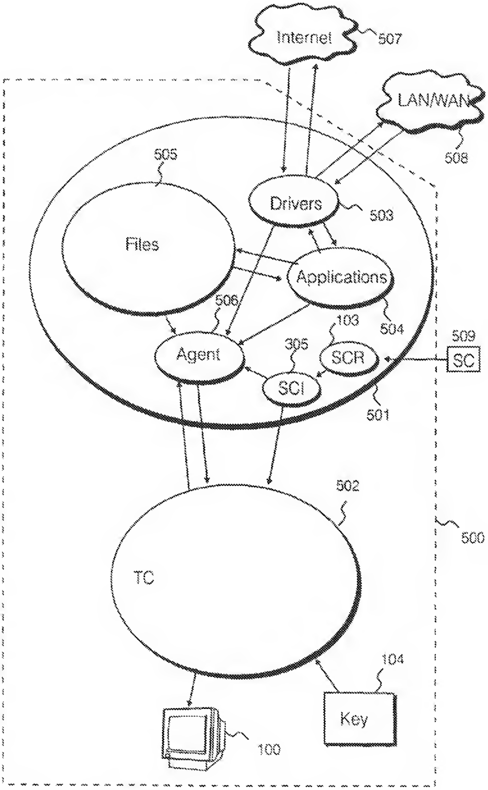

Internet
507
LAN/WAN
508
505
Drivers
503
Files
Applications
506
504
103
305
509
Agent
SCR
SC
SCI
501
502
500
TC
104
Key
100

Fig. 5

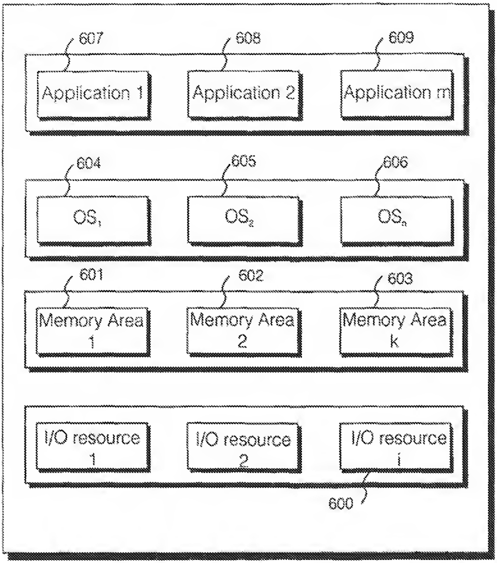

607
608
609
Application 1
Application 2
Application m
604
605
606
OS_1
OS_2
OS_n
601
602
603
Memory Area 1
Memory Area 2
Memory Area k
I/O resource 1
I/O resource 2
I/O resource i
600

Fig. 6

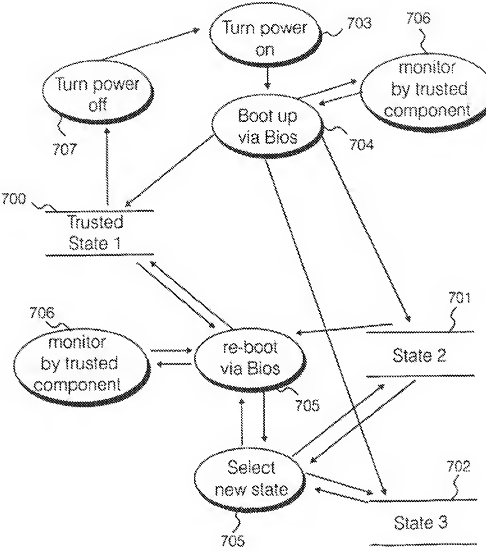

Turn power on
703
706
Turn power off
monitor by trusted component
Boot up via Bios
707
704
700
Trusted State 1
701
706
monitor by trusted component
re-boot via Bios
State 2
705
Select new state
702
State 3
705

Fig. 7

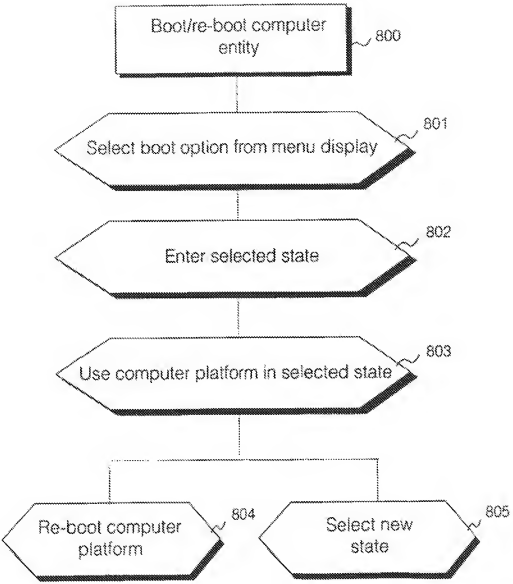

Boot/re-boot computer entity
800
Select boot option from menu display
801
Enter selected state
802
Use computer platform in selected state
803
Re-boot computer platform
804
Select new state
805

Fig. 8

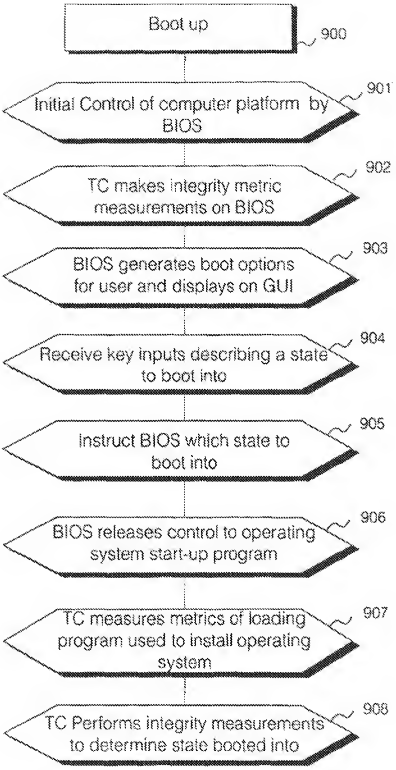

Boot up
900
Initial Control of computer platform by BIOS
901
TC makes integrity metric measurements on BIOS
902
BIOS generates boot options for user and displays on GUI
903
Receive key inputs describing a state to boot into
904
Instruct BIOS which state to boot into
905
BIOS releases control to operating system start-up program
906
TC measures metrics of loading program used to install operating system
907
TC Performs integrity measurements to determine state booted into
908

Fig. 9

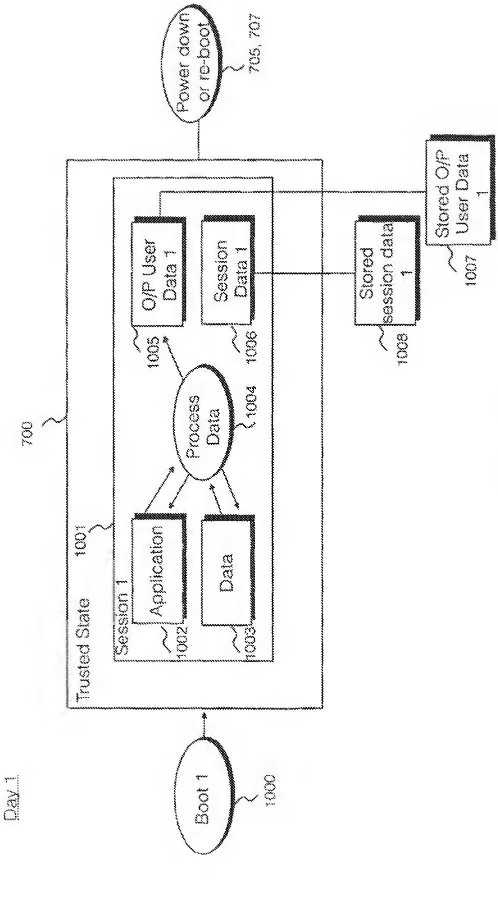

Day 1
Boot 1
1000
700
Trusted State
Session 1
1001
Application
1002
Data
1003
Process Data
1004
O/P User Data 1
1005
Session Data 1
1006
Stored session data 1
1008
Stored O/P User Data 1
1007
Power down or re-boot
705, 707

Fig. 10

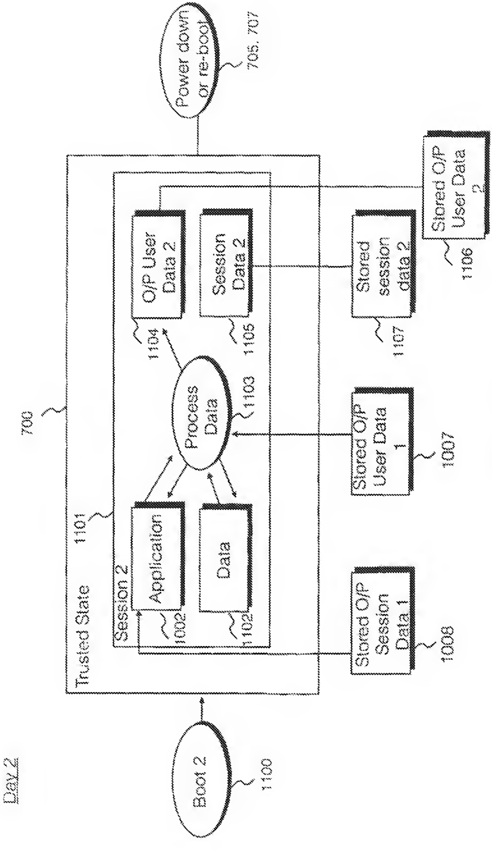

Day 2
Boot 2
1100
Trusted State
700
Session 2
1101
Application
1002
Data
1102
Process Data
1103
O/P User Data 2
1104
Session Data 2
1105
Power down or re-boot
705, 707
Stored O/P Session Data 1
1008
Stored O/P User Data 1
1007
Stored session data 2
1107
Stored O/P User Data 2
1106

Fig. 11

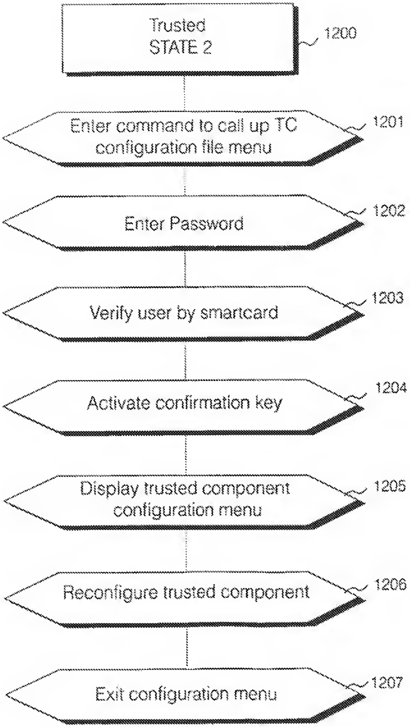

Trusted
STATE 2
1200
Enter command to call up TC configuration file menu
1201
Enter Password
1202
Verify user by smartcard
1203
Activate confirmation key
1204
Display trusted component configuration menu
1205
Reconfigure trusted component
1206
Exit configuration menu
1207

Fig. 12

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 99 30 7380

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.7)
X	WO 95 24696 A (INTEGRATED TECH AMERICA ;MOONEY DAVID M (US); WOOD DAVID E (US); K) 14 September 1995 (1995-09-14) * the whole document *	1,2,4,6,7,18,20-22	G06F1/00
A		3,5,8-17,19	
A	WO 98 15082 A (INTEL CORP) 9 April 1998 (1998-04-09) * abstract; figure 1 * * claims 1-23 *	1-22	
A	EP 0 849 657 A (NCR INT INC) 24 June 1998 (1998-06-24) * abstract; figure 1 * * claims 1-8 *	1-22	
A	CA 2 187 855 A (COMPONENT ORIENTED PROTECTIVE) 13 June 1997 (1997-06-13) * the whole document *	1-22	
A	US 5 680 547 A (CHANG STEVE MING-JANG) 21 October 1997 (1997-10-21) * the whole document *	1-22	TECHNICAL FIELDS SEARCHED (Int.Cl.7) G06F
A	WO 98 36517 A (JPC INC) 20 August 1998 (1998-08-20) * the whole document *	1-22	

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	17 March 2000	Powell, D

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

ANNEX TO THE EUROPEAN SEARCH REPORT ON EUROPEAN PATENT APPLICATION NO.

EP 99 30 7380

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report. The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

17-03-2000

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 9524696 A	14-09-1995	US 5610981 A	11-03-1997
		AT 175505 T	15-01-1999
		AU 703856 B	01-04-1999
		AU 2092695 A	25-09-1995
		BR 9506968 A	01-06-1999
		CA 2183759 A	14-09-1995
		CN 1146813 A	02-04-1997
		DE 69507129 D	18-02-1999
		DE 69507129 T	05-08-1999
		EP 0748474 A	18-12-1996
		NZ 282954 A	24-11-1997
WO 9815082 A	09-04-1998	US 5844986 A	01-12-1998
		AU 4146197 A	24-04-1998
		CN 1231787 A	13-10-1999
		EP 0932953 A	04-08-1999
EP 0849657 A	24-06-1998	JP 10282884 A	23-10-1998
CA 2187855 A	13-06-1997	NONE	
US 5680547 A	21-10-1997	US 5444850 A	22-08-1995
		AU 1042895 A	15-05-1996
		JP 10511783 T	10-11-1998
		WO 9613002 A	02-05-1996
WO 9836517 A	20-08-1998	US 5953502 A	14-09-1999

EPO FORM P0459

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82